OPTIMIZING THE REPLICATION OF
MULTI-QUALITY WEB APPLICATIONS
USING ACO AND WOLF

THESIS

Judson C Dressler, Second Lieutenant, USAF

AFIT/GCS/ENG/06-05

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/06-05

Optimizing the Replication of Multi-Quality Web Applications Using ACO and WoLF

THESIS

Presented to the Faculty
Department of Electrical and Computer Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
In Partial Fulfillment of the Requirements for the
Degree of Master of Science in Computer Systems

Judson C Dressler, B.S.C.S.
Second Lieutenant, USAF

14 September 2006

AFIT/GCS/ENG/06-05

OPTIMIZING THE REPLICATION OF MULTI-QUALITY WEB APPLICATIONS USING ACO AND WOLF

Judson C Dressler, B.S.C.S.

Second Lieutenant, USAF

Approved:

/signed/	14 Aug 2006
Maj Christopher Mayer, PhD (Chairman)	date
/signed/	14 Aug 2006
Dr. Gilbert Peterson (Member)	date
/signed/	14 Aug 2006
Dr. Gary Lamont (Member)	date

AFIT/GCS/ENG/06-05

Abstract

Since its introduction, the Ant Colony Optimization (ACO) meta-heuristic has been successfully applied to a wide range of combinatorial problems. This thesis presents the adaptation of ACO to a new NP-hard problem involving the replication of multi-quality database-driven web applications (DAs) by a large application service provider (ASP). This problem is a special case of the generalized assignment problem (GAP) which occurs in many military contexts such as logistics planning, air crew scheduling, and communications network management.

The ASP must assign DA replicas to its network of heterogeneous servers so that user demand is satisfied at the desired quality level and replica update loads are minimized.

The ACO algorithm proposed, AntDA, for solving the ASP's replication problem is novel in several respects: ants traverse a bipartite graph in both directions as they construct solutions, pheromone is used for traversing from one side of the bipartite graph to the other and back again, heuristic edge values change as ants construct solutions, and ants may sometimes produce infeasible solutions.

Although experiments show that AntDA outperforms several other solution methods, there was room for improvement in the convergence rates of the ants in finding better solutions. Therefore, in an attempt to achieve the goals of faster convergence and better solution values for larger problems, AntDA was combined with the variable-step policy hill-climbing algorithm called Win or Learn Fast (WoLF). In experimentation, the addition of this learning algorithm in AntDA provided for faster convergence and still outperformed the other solution methods. However, as problem complexity rose, AntDA with the WoLF algorithm converged to statistically significant lesser solutions than those found by AntDA, but at a much faster rate.

Acknowledgements

First and foremost, I would like to thank my wife for her loving support throughout this process. I would also like to thank my family for their support, my advisor Major Mayer and the rest of my committee for their hard work and patience.

Judson C Dressler

Table of Contents

Page

List of Figures

List of Tables

OPTIMIZING THE REPLICATION OF MULTI-QUALITY WEB APPLICATIONS USING ACO AND WOLF

I. Introduction

1.1 Motivation

With the rise of the Internet and advances in services desired by users, companies with websites are finding it more economical to *rent* a network from a provider rather than purchasing and maintaining their own. In order to do this, they turn to Application Service Providers (ASPs). This relatively new business partnership introduces interesting technical problems for both the company and the ASP.

This problem is a special case of the generalized assignment problem (GAP) which occurs in many military contexts such as logistics planning, air crew scheduling, and communications network management.

With growth in size and number of users, making content widely available while reducing the load on the web servers becomes a major challenge. Users want the applications and servers they use to be available at all times and with short response times for those requests. A single web server can not handle all of this traffic. Therefore, they create copies of their content, called replicas, and spread them all over the internet using an ASPs expansive server network. In most cases, user requests for an application are received by the nearest ASP where the replicated logic interprets and processes them. Those requests needing information from the back-end databases are passed to the owner's data center for further processing. This combination of an application and its database is referred to as a *database application* or DA for short (see Figure 1).

There would be no problem if the back-end databases data never changed. However, users of most applications require a certain quality of *data freshness*. The quality of data

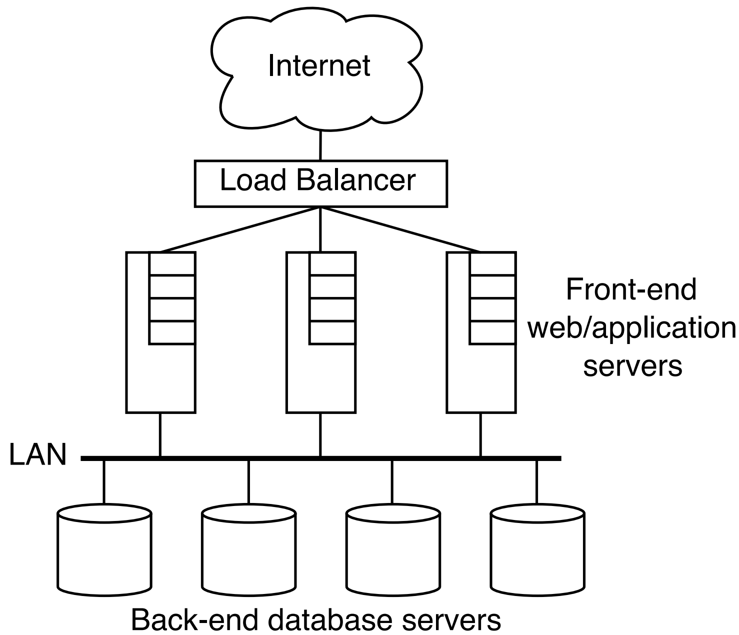

Figure 1: An example e-commerce site.

freshness depends on the freshness of the data in the database and the higher the quality of freshness requested, the more frequently the database needs updating.

Although the quality aspect allows for some flexibility in replicating DAs, it presents some significant issues. For one, user demand for a DA's content is unpredictable and sudden surges in DA demand occur. Second, updating a DA replica with fresh data diminishes the replica's capacity for handling end-user requests. If not managed effectively, this parasitic update load could cause more replicas to be created than needed. Finally, the back-end databases of DA replicas can be huge and take a non-trivial amount of time to move and reestablish replicas.

The core problem dealt with in this thesis investigation is that of an ASP's assignment of replicas of its customers' DAs on its network of servers so as to satisfy user demand (including the appropriate quality) for the DAs while minimizing the parasitic database update load of the DA replicas. This problem is known as the Quality-Sensitive DA Replication Problem, or DA*rep* for short.

1.2 Solution Approach

In order to solve the DA*rep* problem, an ant colony optimization algorithm, AntDA (originally proposed in [53]), is investigated. First, AntDA's parameters are tuned and it is tested against many different test cases. Results show that AntDA performs better than the other search algorithms tested but had higher solution execution times.

In order to minimize the drawback of higher solution times, AntDA is combined with a variable-step policy hill-climbing algorithm called Win or Learn Fast (WoLF). Two different definitions of the WoLF algorithm is experimented with to produce two variations of AntDA: WoLFAntDA and PD-WoLFAntDA. The addition of the WoLF learning algorithm into AntDA allows the ACO heuristic to be applied to more complex problems while still being solved in a reasonable amount of time.

1.3 Thesis Organization

The problem of replicating Web-based applications and copies of their associated databases (DA replicas) that are being updated in order to meet the quality demands of its users requests is the Quality-Sensitive DA Replication Problem (the DA*rep* problem). This thesis effort examines this one aspect of replicating and delivering differing quality Internet content and presents different approaches for solving the problem and effective implementation.

The remainder of this document is organized in the following manner. Chapter 2 delves deeper into the background of the Quality-Sensitive DA Replication Problem. The first part of the chapter describes the challenges of the DA*rep* problem in detail as well as a non-mathematical definition. It also presents a formalized variation of DA*rep* in which servers process updates and requests with varying efficiencies. The following sections provide definitions of other assignment problems as well as an in depth description of the Ant Colony Optimization algorithm as well as the Win or Learn Fast algorithm which were adapted to solve the DA*rep* problem and evaluated experimentally. Chapter 3 provides the insight into how the experiments are conducted with descriptions of how the Ant Colony

Optimization and Win or Learn Fast algorithms were adapted to fit the DA*rep* problem. It also describes briefly three other search algorithms for solving assignment problems that were used for comparison. Chapter 4 contains the initial performance results for AntDA and explains the effects of changes and alternatives to the AntDA algorithm that were made in order to improve performance. It concludes with the performance results of WoLFAntDA and a comparison of its effect on the AntDA algorithm. The conclusion of the main body of this thesis is in Chapter 5. It contains a summary of the main contributions of this thesis and avenues for further research.

II. Background and Related Work

2.1 Introduction

This chapter provides an overview of the background and related work on the Quality-Sensitive DA Replication Problem and Ant Colony Optimization. It starts out with a description of the Quality-Sensitive DA Replication Problem and its environment and discusses other assignment problems related to the Quality-Sensitive DA Replication Problem. The Ant Colony Optimization technique is then explained, followed by an in depth review of the Win or Learn Fast algorithm.

2.2 Problem Background

This section presents a deeper introduction to the environment of the Quality-Sensitive DA Replication Problem and a non-mathematical problem statement. The problem is then formulated mathematically as a 0-1 assignment problem [53] and is proven to be NP-hard even when each DA in the system has only one quality-level

*2.2.1 The DA*rep *Environment.* With the rise of the internet, web-based applications have become very complicated. The applications and services are now implemented as a combination of application logic that takes user requests and talks to the back-end databases to acquire the data content necessary to generate the appropriate response. These systems are referred to as Database Applications, or DAs for short, and are commonly seen in e-commerce and e-business (on-line stores, auctions, news services, banking, etc.) (see the Fig. 1 on page 2).[1] However, as Table 1 shows this web-centric model of DAs is not fundamentally different from that found in other contexts such as scientific grid computing [4, 5, 39, 44] and data warehousing [67, 68].

With the explosion of internet users in the late 1990's and today, some database applications have become too popular for owner's to be able to keep up the infrastructure necessary to handle the traffic. Consequently, these owners have turned to application service

[1] Notionally a *server* is a single computer or a group of computers working in concert to implement a DA or a DA replica. To simplify matters, this document hides this multiple-computer notion of *server* by always assuming a server is a single computer.

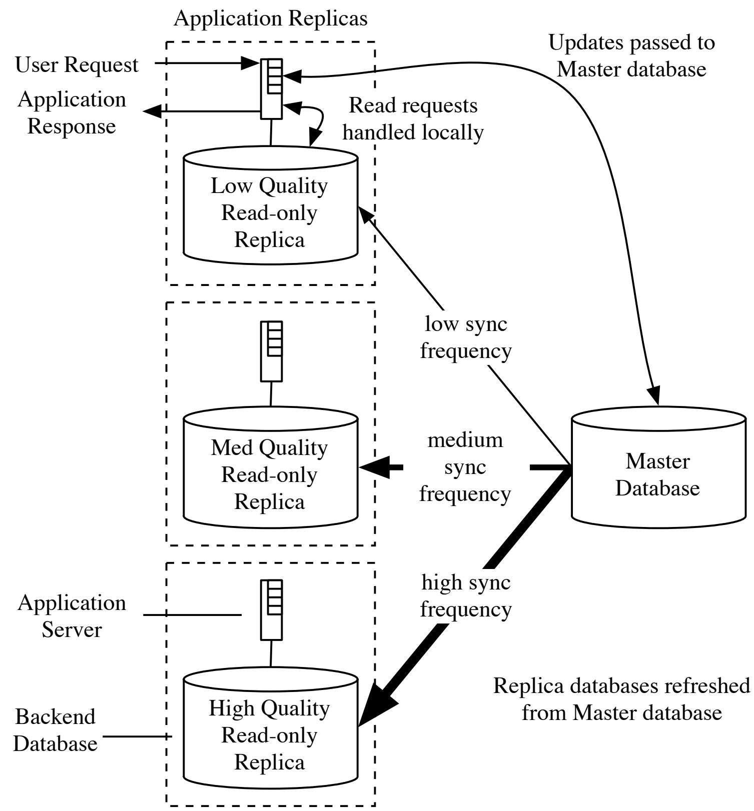

Figure 2: A Typical Application and Database (DA) Replica Setup. DA replicas consist of an application servers and local read-only database replicas. When reading data to fulfill a user request, an application server accesses its local database. When generating or changing data, the master database is contacted. The frequency at which the master database synchronizes its database replicas determines the service quality provided by the replica.[53]

providers (ASPs), such as Akamai [2] and ASP-One [6], in order to relieve some of their application's workload by distributing the application logic onto the ASP's extensive network of servers. This also relieves the owner of handling infrastructure needs such as hardware, network, backup, security, and operating systems, making the owner's process much more manageable. In most cases, requests for the application are received by the ASP where the replicated logic interprets and processes them. Those requests needing information from the back-end databases are passed to the owner's data center for further processing. By doing this, access to the database often becomes the major performance bottleneck of the process [65]. Therefore, replicating just the application logic may be insufficient. *In order for an owner to receive maximum benefit, the database must be replicated too* [48, 58].

	Description		
Similarity	Web-based DA	Data Warehousing	Scientific Grid Computing
Goal	Assign DA replicas to servers, direct requests to appropriate replica, avoid DBMS overload	Place DW replicas on servers, data mine on appropriate replicas, avoid DBMS overload	Assign compute tasks to servers avoid DBMS overload at servers
Servers	Hundreds of servers each with its own capacity limit	Hundreds of servers each with its own capacity limit	Hundreds of servers each with its own capacity limit
Bottleneck	DBMS	DBMS	Computing tasks and DBMS
Data	Large databases generate HTML	Large Databases store warehouse data	Experiment data in massive Databases
Load Sources	User requests and replica database synchronization	Data mining operations and new data added to DW	Tasks query database and generate new data.
Service Qualities	Data freshness requirements of users create quality levels	Levels: managers (low) and data analysts (high)	Levels: making hypothesis (low) and proving hypothesis (high)

Table 1: DAs, Data Warehousing, and Grid Computing Compared [53].

Recent advances in database caching and update propagation have made replicating the database portion of multi-tiered web application (Fig. 2) more feasible [17, 18, 21, 28, 47, 49, 51, 55]. Nevertheless, database replication still has many issues. Most importantly, the database replicas must be periodically updated so that their content is timely or *fresh*. Updating a database replica with fresh data, strips the replica of capacity for handling end-user request and may cause the need for many more replicas. Also, databases are normally many gigabytes in size and can take a great deal of time to move and establish replicas.

One of the main difficulties faced by an application service provider (ASP) is the decision of where to assign replicas of its customers' DAs on its network of servers. In doing this, it must consider how to best meet user demand for the DAs and keep the cost of database updates for the DA replicas minimized. Users having differing expectations about the timeliness or freshness of the content they receive severely complicate this assignment problem [13, 22]. In other words, *users may have service quality requirements that have to be met*. The implication of this is two-fold. First, not every DA replica must operate at the highest quality level as there may be users which are happy with a lower quality. Secondly, a request must be served from a replica (a copy of the application logic and

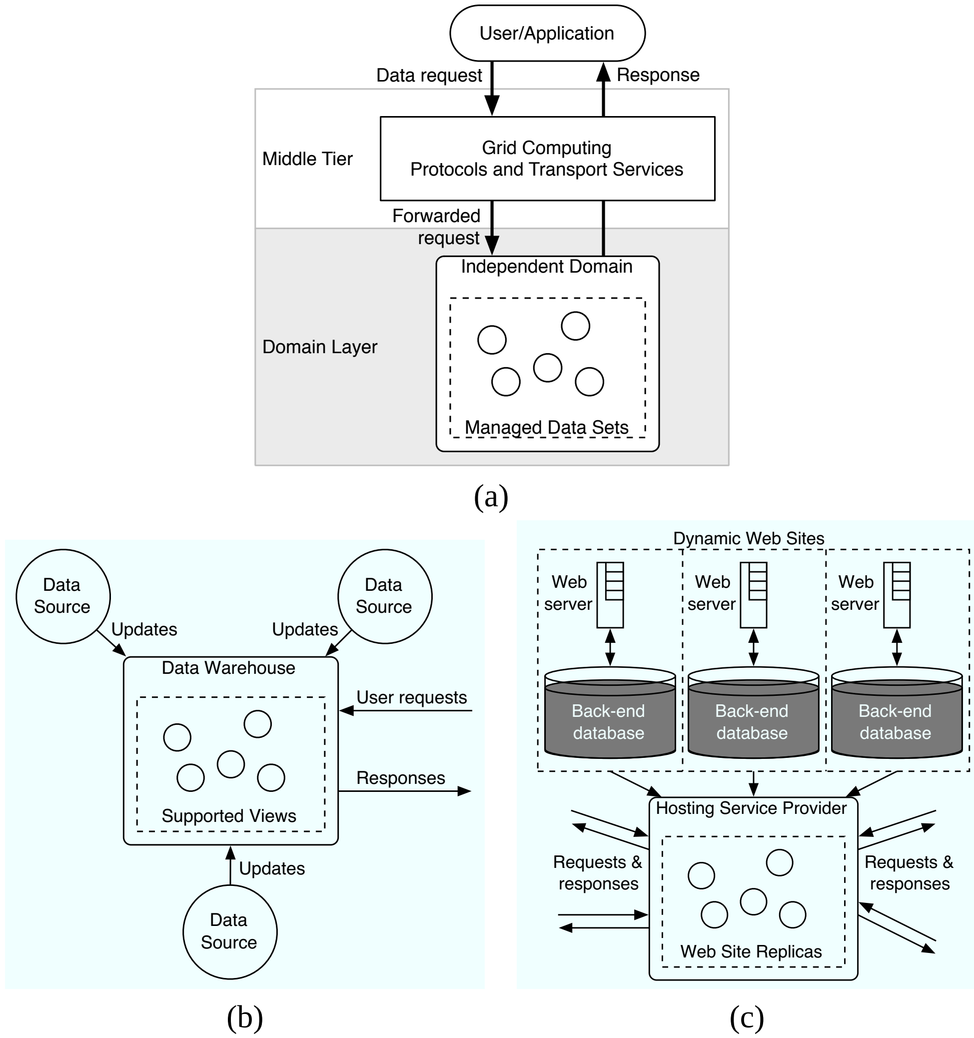

Figure 3: Distributed Environments Requiring Effective Replication Solutions. Shown are examples of (a) Grid computing domains, (b) data warehousing operations, (c) application service providers replicating dynamic Web sites [53].

relevant portions of the database) that meets or exceeds the user's quality requirement. The ASP's problem of deciding on replica-to-server assignments of such quality-differentiated DAs is the **Quality-Sensitive DA Replication Problem**, or DA*rep* for short.

2.2.2 Quality-Sensitive DA Replication Problem Assignment Issues. For the Quality-Sensitive DA Replication Problem (DA*rep*), there are several issues that must be taken into account when assigning replicas to servers: application masters and replica

slaves, dynamic content, keeping replicas fresh, large databases, and DBMS load and replica server response times.

Application Masters and Replica Slaves: A trend in DA architecture and replication is the user of the master/slave relationship for updating the database application. In operation, there is a single master and zero or more slaves (or replicas). Both masters and slaves contain a database component and are capable of receiving and handling user requests. When a slave needs to update its database, it contacts the master which processes the update and propagates the appropriate changes to the slaves, thereby refreshing their databases. Oracle's Database Cache and IBM/DB2 support master/slave replication [19].

Dynamic Content: While improvements in dynamic content caching have been made and techniques exist to synch masters and slaves [17,18,21,28,47,49,51,55], caching's benefits are limited, meaning that ample access to source data is always required. Replication can provide this ample access. An application and a relevant portion of its database can be replicated by a service provider as is done by Akamai using IBM's WebSphere product [42]. Figure 2 shows the replication of a single DA. An important concept to remember is that updates occur at a master database which propagates changes to database replicas based on the replicas' freshness requirements.

Keeping Replicas Fresh: Database replicas have to be regularly updated so that their content is timely or *fresh*. Assigning a DA replica to a server induces a continuous update load on the server's database component due to the frequent updates required to maintain the replica's service quality. This update load is parasitic in the sense that it reduces the replica's capacity for handling end-user requests. This resource drain also excludes plans of creating more replicas than demand warrants and limits how many replicas a server can host. A higher quality of service requires fresher data and require more frequent database synchronization. Therefore, update load on a replica increases with freshness or service quality. Understandably, update load mitigation has been the subject of much research [17,18,20,46,49,52,57].

Large Databases: The database of a DA is normally many gigabytes large and can take a great deal of time to move and establish replicas in response to changing demand.

DBMS Load and Replica Server Response Times: Response times are the crucial measurement of speed in today's internet, especially for e-commerce applications, since slow response times translate into unhappy customers and lost revenue [47, 65]. Since request and response sizes are small and propagate quickly over today's internet, the performance bottleneck for database-driven applications has been shown to be the Database Management System (DBMS). DA response times depend greatly on database load, and not necessarily the placement of the replicas geographically close to users or network delays [18,28,47,65]. The database load of a DA replica has two components: request load and update load. Request load results from queries stemming from user requests. Update load is the resources consumed in synchronizing a replica's slave database with its master database. Therefore, if the DBMS can keep the request and update loads from overloading the database, response times will be managable.

2.2.3 A Non-mathematical Definition of the DA Problem. The concepts presented up to this point are used to define the **Quality-Sensitive DA Replication Problem**, DA*rep* as shown in Fig. 4. All terms used in the definition were defined previously in this chapter.

Figure 5 contains an ASP replication scenario that fits the DA*rep* definition. Customers maintain *master databases* from which updates are disseminated. Once replicas are assigned to the ASP's servers, the database replicas are synchronized with customers' master databases so as to maintain each replica's designated service quality (Figs. 2 and 5). Since users access replicas, which are read-only, any request that changes a DA's database is routed through the master database and then disseminated to the replicas. Replica update/synchronization is costly in terms of resources, and hence, has to be minimized while maintaining the appropriate freshness levels.

This thesis presents and evaluates several algorithms for solving various forms of Quality-Sensitive DA Replication Problem.

The Quality-Sensitive DA Replication Problem

Given

- A set of DAs to be replicated where each DA has one or more freshness quality levels at which it operates.
- A set of servers on which DAs can be placed. Each server has a (possibly different) load limit. Zero or more DAs can be hosted on server.
- Request rates for DA content and freshness levels for each web site to be replicated.
- Operating loads:
  - Update load: The load of maintaining a DA replica at a certain freshness level is the product of an update rate and DA update complexity.
  - Request load: The load for handling requests for a DA replica is the product of the total request rate at the replica and the expected request complexity for the DA.
- Requests are considered satisfied only if the returned content meets or exceeds a minimal freshness requirement stated in the request.

Find an assignment of

1. DA replicas to servers,
2. freshness quality levels to DA replicas, and
3. a distribution of requests to replicas that fulfills the application and quality demands of the requests

such that, for each server, the sum of the update and request loads for DA replicas hosted by a server does not exceed the server's load limit.

Figure 4: The Quality-Sensitive DA Replication Problem Informally Defined

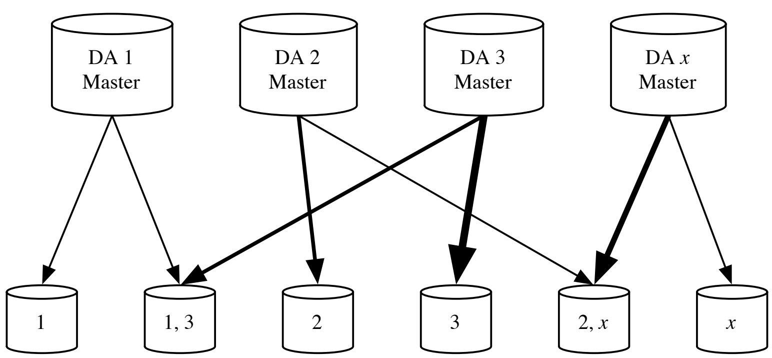

Figure 5: An Application Service Provider Network Hosting DAs [53].

2.2.4 The Quality-Sensitive DA Replication Problem Formalized. In this subsection, the DA*rep* problem is formulated as a linear, mixed-integer minimization problem which has been proven to be NP complete [53].

Definitions:

- The ASP has m servers, $S = \{1, \ldots, m\}$.
- Each server $s \in S$ has a processing capacity denoted by C_s.
- The ASP has n customer-provided DAs to be hosted, $D = \{1, \ldots, n\}$, on its m servers.
- Each $d \in D$ operates at one or more service quality levels, $Q_d = \{1, \ldots, q, \ldots, qmax(d)\}$, where $qmax(d)$ is the highest level offered by DA d.
- The request load for DA d, RL_d, is the sum of the request loads, denoted by rl_d, for each of its quality levels: $RL_d = \sum_{q \in Q_d} rl_{d,q}$.
- For each service quality of a DA d there is a certain update load required to maintain that service quality: $UL_d = \{ul_{d,1}, \ldots, l_{d,q}, \ldots, ul_{d,qmax(d)}\}$.
- Let $x_{s,d,q} \in \{0, 1\}$ be a binary variable that indicates that server s is hosting a replica of a certain $\langle d, q \rangle$ pair where $\langle d, q \rangle$ *pair* is shorthand for *quality* q *of DA* d. Let $\lambda_{s,d,q} \in [0, 1]$ denote the fraction of the request load of a $\langle d, q \rangle$ pair, $rl_{d,q}$, assigned to server s. The update load experienced by server s depends on the quality level of the DA replicas it hosts:

  $$ul_s = \sum_{d \in D} \sum_{q \in Q_d} x_{s,d,q} \cdot ul_{d,q}. \tag{1}$$

  Server s's request load is the fraction of each $\langle d, q \rangle$ pair's request load sent to it:

  $$rl_s = \sum_{d \in D} \sum_{q \in Q_d} \lambda_{s,d,q} \cdot rl_{d,q}. \tag{2}$$

Objective and Constraints: The ASP seeks an assignment of DA replicas to servers that minimizes the system-wide update burden, UB, and is subject to four constraints.

$$\min\ UB = \min \sum_{s \in S} \sum_{d \in D} \sum_{q \in Q_d} ul_{d,q} \cdot x_{s,d,q} \tag{3}$$

1. Request load for each quality of each DA is satisfied: $\sum_{q \in Q_d} \sum_{s \in S} \lambda_{s,d,q} = 1$.

2. Only one quality level of a DA is hosted by a server: $\sum_{q \in Q_d} x_{s,d,q} \leq 1$.

3. A server's processing capacity cannot be exceeded: $ul_s + rl_s \leq C_s$.

4. Requests processed by a replica must meet or exceed the request's quality expectation, q_r: $\sum_{q_s | q_s, q_r \in Q_d \wedge q_s \geq q_s} x_{s,d,q_s} \geq \lambda_{s,d,q_r}$.

Although, in theory, the above formulation could enable the ASP to find optimal DA assignments, in reality, optimal solutions are elusive since DA*rep* is in the class of NP-hard problems, even if all the DAs in an ASP have only one freshness quality level, as shown in [53].

2.3 Other Assignment Problems

Essentially, the Quality-Sensitive DA Replication Problem is an optimization assignment problem with many constraints. This section covers a couple of the categorical assignment problems and how they have been solved.

Pairing problems constitute a vast family of problems which deal with practical design and resource-allocation problems. Different versions of these problems have been studied since the mid 1950s due both to their many applications and to the challenge of understanding their combinatorial nature. Some can be easily solved in polynomial time, whereas others are extremely difficult. The simplest one is the Assignment Problem that can be easily solved by the Hungarian Algorithm [61]. Others are much harder, such as the Generalized Assignment Problem and the Quadratic Assignment Problem, which are very difficult and NP hard [61].

2.3.1 Generalized Assignment Problem. Assignment problems consist of finding the best assignment of some set of items to items (or agent) of another disjoint set according to some predefined function. Its many applications include the assignment of tasks to workers, of jobs to machines, of fleets of aircraft to tasking orders, or the assignment of school buses to routes [1, 60]. However, in most practical applications, each agent requires a quantity of some limited resource to process a given job or has a limited capacity for a given resource. Therefore, the assignments have to be made taking into account

the resource necessity or capacity of each agent. The problem derived from the classical Assignment Problem by taking into account these capacity constraints is known as the Generalized Assignment Problem (GAP). Among its many applications is the problem of assigning variable length commercials to time slots [7], jobs to computers in a computer network [7], distribution of activities to the different subsections of a company when making a project plan [69], etc. Besides these applications, it also appears as a subproblem in a variety of combinatorial problems like Vehicle Routing [36] or Resource Location [31, 61]. A classic NP-hard problem that is similar to the Generalized Assignment Problem is graph coloring [24, 25].

2.3.2 Quadratic Assignment Problem. The Quadratic Assignment Problem (QAP) is another classic combinatorial optimization problem and is widely regarded as one of the most difficult problems in this class. It was first introduced by Koopmans and Beckman to solve a facilities location problem [59]. The problem involves assigning N facilities to N locations so that the cost of the assignment, Z, is minimal. It can be defined as follows.

$$\begin{aligned}
&\min Z = \sum_{\mathrm{i}=1}^{N}\sum_{j=1}^{N} a_{ij}x_{ij} + \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N}\sum_{l=1}^{N} f_{ik}c_{jl}x_{ij}x_{kl} \\
&s.t. \\
&\sum_{j=1}^{N} x_{ij} = 1, i = 1, 2, \ldots, N \\
&\sum_{i=1}^{N} x_{ij} = 1, j = 1, 2, \ldots, N \\
&x_{ij} \in \{0, 1\}, i, j = 1, 2, \ldots, N
\end{aligned} \tag{4}$$

where N = total number of facilities, a_{ij} = fixed cost of locating facility i at location j, f_{ik} = flow of material from facility i to facility k, c_{jl} = cost of transferring a material unit from location j to location l, and x_{ij} = 1, if facility i is at location j; 0 otherwise.

Many solution methods have been developed to address the QAP because of its considerable practical importance in facility layout, machine scheduling and other applications. Even with fast computers, exact algorithms such as branch-and-bound methods are only able to globally solve rather small QAPs in a reasonable amount of time [34]. Therefore,

researchers have concentrated on developing effective heuristics for the QAP. The diverse QAP-heuristics are not examined here. Instead, extensive assessments of the QAP and its associated solution methods can be found in [16, 34, 45]. Recent developments in facility layout are also covered in [40, 41]. Many classic NP-hard problems fall into the QAP category such as bin-packing and the knapsack problem [34]. The DA*rep* also resides in the QAP category of assignment problems [53].

2.4 Ant Colony Optimization

Despite the current technology, and rapid advances in every field, there are still some problems that continue to elude scientists. Learning algorithms have been developed in combination with artificial intelligence systems such as neural networks to try and solve some of these problems, but imperfections and inefficiencies in both the hardware and software often prevent reliable results. Scientists, are now looking into the world of insects, or swarm intelligence for inspiration for new methods and approaches of attacking complex problems. This section describes how one particular form of swarm intelligence, the Ant Colony Optimization (ACO) meta-heuristic algorithm uses the cooperative nature of ants in order to solve difficult combinatorial optimization problems.

2.4.1 Ant Algorithm Background. An individual ant is relatively unintelligent, but as a part of a colony, a complex group behavior emerges from the interactions of individuals who exhibit simple behaviors by themselves [34, 53]. This phenomenon is indicative of all swarm intelligences, where something is created that is greater than the sum of its parts. Using their social structure ants are able to complete very complex tasks without even knowing of the existence of the problem [34, 53]. One of the complex behaviors that naturally emerges from the ant colony is the ability to determine the shortest path between two points. An important insight of ant behavior is that most communication among individuals, or between individuals, is based on the use of chemicals, called pheromones, produced by the ants. Particularly important for the social life of ants is the trail pheromone, a pheromone that individuals deposit while walking in search of food [53]. By sensing the

level of pheromone on trails, forager ants can follow the path found by other ants to get to food. The behavior of pheromone-laying and pheromone following is the inspiring source of ant colony optimization. Initially, all ants move randomly from their starting point in search of food, since there is no pheromone to start with, all ants choose between paths with equal probability. While walking, ants deposit on the ground a pheromone trail; when choosing which way to go when their trail forks, ants choose with higher probability those directions marked by a stronger pheromone concentration [34, 53]. However, ants sometimes behave randomly and select trails with lighter concentrations or even investigate a new trail altogether. This random behavior promotes the exploration and discovery of other paths which enhances the chances of finding the best solution possible. An ant continues to follow trails until it reaches its goal or gets tired. Either way, each ant will return to the nest while laying pheromone. The concentration of the pheromone trail is directly proportional to the impact of the goal found. For example, if the food item is highly appetizing and could not be taken by the single ant, then a large amount of pheromone would be deposited on the ants return trip to make sure other ants would find their way to it. Likewise, if the food item is not appetizing, is small in quantity, or nothing was found at all, the ant would deposit less pheromone which would make other ants not pay as much attention to that trail. Since pheromone evaporates over time, trails leading to a big reward are continually reinforced, while trails leading to little or no reward fade away [53].

When choosing between a shorter and longer trail leading to the same goal, those ants choosing the shorter branch find the food first and are first to get back to the nest. Therefore, more ants traverse the shorter branch and the pheromone trail on this branch will grow faster, thus increasing the probability that it is used by approaching ants. This process of positive feedback is at the heart of the ant colony behavior that can very quickly lead all the ants to choosing the shortest branch. This behavior has been adapted into an algorithm which can be used by artificial ants to find minimum cost paths on graphs, as explained in the next section.

2.4.2 The Ant Colony Meta-heuristic. A colony of (artificial) ants traverse a graph where the graph's edges (directed or undirected) can be seen as the ants possible trails and the graph's vertices as decision points. While traversing the graph, the ant records its path taken and remembers its cost. Each ant's path is a solution to the problem and the more desirable the cost, the better the solution to the problem. After each iteration of finding a solution, each ant deposits pheromone on the edges it traversed. The amount of pheromone laid by each ant depends on the desirability of its solution cost, usually done by adding pheromone equal to the inverse of the solution cost (eg. Solution cost = 800, pheromone laid = 1/800). In other words, edges used to find the best solutions (least cost) are reinforced with more pheromone than edges that are determined to lead to worse solutions. With time, the pheromone on the edges evaporates, making undesirable edges less attractive over time.

At every vertex, the ant observes the pheromone levels of all outgoing edges of that vertex. It then, based on each outgoing edge's pheromone concentration and a heuristic desirability, makes a probabilistic choice about which edge to follow. The ants use both pheromone (representing past good solutions) and a heuristic value (to guide ants when little is known about an edge's desirability) to encourage the exploration of solutions in the region of known good solutions, but is still random enough that good solutions are highly unlikely to go undiscovered [53].

Many NP-complete combinatorial problems have been attempted with ant algorithms. These include the traveling salesman problem [14, 33, 35], job-shop scheduling [23], graph coloring [24, 25], vehicle routing [15, 56], adaptive routing in communication networks [10, 29, 30, 63, 66], sequential ordering [38], shortest common supersequence [54], and multidimensional knapsack [3]. In each case, ant algorithms perform as well or better than the best known algorithms for solving the problems above as a majority of the works cited above indicates.

Typically, ant algorithms consist of a doubly nested loop. The outer loop controls the number of iterations (usually called time steps) executed and the inner loop controls each ant as it traverses the graph and builds a solution. Once each ant has completed building

its solution, the pheromone on the graph's edges are updated to reflect the quality of the solutions found in the current time step and account for the evaporation of pheromone from infrequently-used edges. This updated graph is then used as the starting graph for the next iteration.

According to Ant Colony Optimization Pioneers - Eric Bonabeau, Marco Dorigo, and Guy Theraulaz, there are four essential elements to an ant algorithm. This next section is adapted from [9].

1. **Heuristic Desirability.** This element gives the desirability of moving from vertex i to vertex j based only on local information. Heuristic desirability is denoted by η_{ij}. Since this element is problem specific, no definitive or typical equation can be given. However, for example's sake, at least two TSP algorithms use the inverse of inter-city distances as the heuristic. Thus, cities that are closer together are more attractive than cities that are farther apart.

2. **Transition Rule.** This rule determines the probability that an ant k follows edge (i, j) when moving from vertex i to vertex j. Let J_i^k be the set of vertices ant k can move to if currently at vertex i. A typical rule is:

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}]^\beta}{\sum\limits_{l \in J_i^k} [\tau_{il}(t)]^\alpha \cdot [\eta_{il}]^\beta} \tag{5}$$

   when $j \in J_i^k$ and 0 when $j \notin J_i^k$. In the above equation, $\tau_{ij}(t)$ is the pheromone concentration on edge (i, j) at time step t. Scaling parameters α and β control the influence of pheromone trail $\tau_{ij}(t)$ and heuristic desirability, η_{ij}, respectively. Note that $\sum\limits_{j \in J_i^k} p_{ij}^k = 1$ as long as J_i^k is non-empty.

3. **Constraint Satisfaction.** This element ensures that the solutions found are feasible. For example, maintaining a list of visited cities ensures that ants visit each city exactly once during a tour in the TSP.

4. **Pheromone Update Rule.** This rule governs the updating of pheromone on edges. Pheromone is deposited on edges to reflect the quality of solutions found by the

ants. Evaporation of pheromone from the edges also occurs. In ant TSP algorithms the concentration of pheromone deposited on edges is inversely proportional to the shortness of a tour or set of tours, i.e., edges making up short tours receive more pheromone than longer tours. Evaporation of pheromone is typically modelled as a constant phenomenon; each time step a constant fraction of pheromone evaporates from all edges. A typical update rule is:

$$\tau_{ij}(t+1) \leftarrow (1-\rho) \cdot \tau_{ij}(t) + \rho \cdot \Delta\tau_{ij}(t) \tag{6}$$

where $\tau_{ij}(t)$ is the amount of pheromone on edge (i, j) at time step t, $\Delta\tau_{ij}(t)$ is the amount of new pheromone to be deposited on edge (i, j) as a result of the ants' collective activity during time step t, and ρ determines the fraction of old pheromone to new pheromone. Note that the determination of $\Delta\tau_{ij}(t)$ is implementation specific. Past researchers have experimented with $\Delta\tau_{ij}(t)$ expressions that (i) rank the solutions of the k ants and deposit pheromone on the edges of the top m, $1 \leq m \leq k$, solutions proportional to each solution's rank [9], (ii) limit the maximum and minimum amount of pheromone on any edge, and (iii) proportionally reinforce stronger pheromone trails less than weaker ones.

Pheromone trails can also be updated dynamically as the ants work. For example, [32] updates edges each on each ant's passing using:

$$\tau_{ij}(t) \leftarrow (1-\rho) \cdot \tau_{ij}(t) + \rho \cdot \tau_0 \tag{7}$$

where τ_0 is a constant amount of pheromone. This style of dynamic updating reduces pheromone on visited edges and encourages exploration of non-visited edges by ants working later in the current time step.

The Ant Colony Optimization algorithm can be adapted to solve many types of optimization algorithms. In section 3.1, it discusses how ACO was adapted in order to solve

the Quality-Sensitive DA Replication Problem problem and in section 4.4, how this optimization technique compares to other methods for solving this problem.

2.5 Reinforcement Learning

The Win or Learn Fast Algorithm is a reinforcement learner. Since this thesis investigation centers on a combination of ACO and WoLF, this section reviews reinforcement learning concepts. It begins with an introduction to learning algorithms and their uses. It discusses policy hill-climbing algorithms and an in depth look at the Win or Learn Fast algorithm and how it has been applied to problems and its effect.

2.5.1 Reinforcement Learning. Reinforcement learning concerns an agent who must learn behavior through trial-and-error interactions with a dynamic environment. The agent's job is to find a policy π, mapping states to actions, that maximizes some long-run measure of performance [43]. However, in order to find the policy, the agent must first explore the problem space and determine, at each state, the effect of choosing each action as it impacts the long-term goal of finding the optimal solution. The following example demonstrates the problem of exploration versus exploitation. The simplest possible reinforcement-learner problem is known as the k-armed bandit problem [64]. In this problem, an agent must pull one of k arms (gambling machines) at each time step so as to maximize the total average reward. The agent is permitted a fixed number of pulls, h. Any arm may be pulled on each turn. The machines do not require a deposit to play; the only cost is wasting a pull. When arm i is pulled, machine i pays off 0 or 1, according to some underlying probability parameter p_i, where payoffs are independent events and the p_is are unknown [43]. The goal of this problem is to determine the best strategy for the agent to take in order to obtain the maximum possible payoff.

This problem illustrates the fundamental tradeoff between exploitation and exploration. An agent who believes that a particular machine has a fairly high payoff probability could choose that arm every time, but it could be missing out on a better probability of winning on another machine. The solution to this problem depends on the number of pulls

allowed or how long the agent is expected to play the game. The longer the game lasts, the worse the consequences of prematurely converging on a sub-optimal machine, and the more the agent should explore before converging on a solution [43].

There are two main strategies for solving reinforcement-learning problems. The first is to search in the problem space in order to find a behavior that performs well in the problem environment. This approach has been taken by those working in genetic algorithms and genetic programming, as well as some novel search techniques [62]. The second uses statistical techniques and dynamic programming methods to estimate the utility of taking actions in states in the world and then choosing the best action based on the statistics generated. This second approach is the basis for the Win or Learn Fast algorithm which is examined in section 2.5.2.

2.5.2 Win or Learn Fast Algorithm. In most learning algorithms involving agents, the solution space is seen as a collection of state-action pairs, represented by (s, a) where $s \in S$ and $a \in A$. The set of states, S, is the particular locations/places where an agent can be located (e.g. the states of DA*rep* are the servers/$\langle d, q \rangle$ pairs). A is the set of all possible actions or moves an agent is allowed to do when in a certain state (e.g. pick a server to host the $\langle d, q \rangle$ pair selected). In policy hill-climbing (PHC) algorithms, each (s, a) pair is given a policy value in the agent's problem space in order to guide an agents decision making toward maximizing the reward (or minimizing the cost) of the problem being solved. The policy of each (s, a) pair, π_{sa}, is updated based on a probability that the action a taken from state s will lead to a better solution [26]. Actions with a high policy value are considered more important to producing optimal results and are more likely to be exploited by the agent in the future [11].

Using the policy hill-climbing algorithm, an agent must explore the solution space, then based on the reward (or lack thereof) received by performing action a in state s, adjusts π_{sa}. Seeing the maximum reward as getting to the top of a hill, the algorithm *climbs* toward the best reward. The policies are adjusted by an amount, δ, which is referred to as the *learning rate* or *step size*.

Normally, PHC algorithms use a single fixed step size, which for several reasons, is not ideal. First, a single fixed step size prevents the algorithm from increasing policies by a larger or smaller amount than the fixed size when necessary. Second, it has been shown that in fixed step size hill-climbing algorithms, an agent's policies never reaches a steady state, or converge for the general problem case [26].

WoLF (Win or Learn Fast) is a policy hill-climbing method by Bowling and Veloso for changing the learning rate to encourage convergence in a multiagent reinforcement learning scenario [12]. WoLF's technique is very intuitive. It suggests that an agent should adapt quickly when doing more poorly than expected, and be cautious when it is doing better than expected so as not to overstep a better strategy. This approach allows for convergence in an agent's policies [12].

The novelty of WoLF is that it replaces the usual single fixed step size, with two learning rates, for each state-action pair. The two step sizes are associated with the concept of *winning* and *losing*.

In WoLF, winning is when the policy for a state action pair is interpreted as leading to an optimal solution. For state action pairs that are considered winning, a small step size, δ_w, is used to update π to encourage exploration in the solution space around the winning state-action pair [12]. However, if a state action pair is losing, it has been shown to lead to a far from optimal solution. Therefore, a large step size, δ_l, is used to dramatically increase the π of a losing state-action pair. This step size allows the WoLF algorithm to exploit recent performance gains uncovered by the losing state-action pair and to move more quickly towards a solution of optimal value. This is what is meant by *learn fast* [26].

The impact an action has is determined by combining the concept of policy with search algorithms such as Policy Hill Climbing [27], Gradient Descent [11], Q-Learning [11, 37], or Ant Colony Optimization [26]. When combined with these algorithms, WoLF uses the strength of an adaptive decision policy to enable agents to converge more rapidly to optimal solutions, thus making these algorithms more effective.

There are three main options when using the Win or Learn Fast algorithm that must be manipulated to fit the problem at hand.

1. **An estimation policy.** This is the value derived from the optimization algorithms discussed. It is the approximation of an agents perceived environment at each state. For gradient ascent algorithms, this would be the probabilities of an action being selected and the expected payoff that would result. In Q-learning, the Q-value can be used to determine an approximation of an agents current environment. For ACO, an edge's pheromone concentration can be used to estimate the ant colony's best guess at which edges should be included in the optimal solution [26].

2. **A rule for determining winning or losing.** This rule determines whether or not an agent is approaching a local optima or not. In [11], Bowling and Veloso introduced an algorithm, called WoLF-PHC, that combined the WoLF concept with the policy hill-climbing variant of Q-learning. In accordance with WoLF, they proposed that the value of δ, whether the (s, a) pair is winning or losing, to be determined by:

$$\delta = \begin{cases} \delta_w, \text{ if } \sum_{a'} \pi_{(s,a')} Q_{(s,a')} > \sum_{a'} \overline{\pi}_{(s,a')} Q_{(s,a')} \\ \delta_l, \text{ otherwise} \end{cases} \tag{8}$$

   where a' are the actions available from state s, $Q_{(s,a')}$ is an (s, a) pairs Q-value, and $\bar{\pi}_{(s,a)}$ is the average of all $\pi_{(s,a')}$ values. Using this equation, an agent is winning if its current policies for all actions in state s have a greater benefit than using the average policy of all actions in state s [11].

   However, Banerjee and Peng, in their Policy Dynamics-Based Win or Learn Fast Policy Hill-Climbing (PDWoLF-PHC) algorithm [8], suggest an alternate definition of winning and losing using the gradient of the policy. This definition relies on keeping track of policy rate of change (policy velocity) $\Delta_{(s,a)}$, and policy acceleration, $\delta\Delta_{(s,a)}$:

$$\delta = \begin{cases} \delta_w, \text{ if } \Delta\pi_{(s,a)}(t) \cdot \Delta\pi^2_{(s,a)}(t) < 0 \\ \delta_l, \text{ otherwise} \end{cases} \tag{9}$$

where

$$\Delta\pi_{(s,a)}(t) = \pi_{(s,a)}(t) - \pi_{(s,a)}(t-1) \tag{10}$$

and

$$\Delta\pi^2_{(s,a)}(t) = \Delta\pi_{(s,a)}(t) - \Delta\pi_{(s,a)}(t-1) \tag{11}$$

According to this definition, the policy for an (s, a) pair is *winning* if:

(a) the policy value is increasing (positive $\Delta\pi_{(s,a)}$) but the rate of increase is slowing down (negative $\Delta\pi^2_{(s,a)}$) or

(b) the policy value is decreasing (negative $\Delta\pi_{(s,a)}$) but the rate of decrease is slowing down (positive $\Delta\pi^2_{(s,a)}$).

This definition uses the fact that policy value change rates should slow down as they near their optimums (from either the positive or negative side). Therefore, when the change rate slows, the (s, a) pair is seen as winning. Otherwise, the edge is seen as losing and needs to take larger step sizes (learn faster) in order to reach its optimal value faster. This definition has been shown to converge more rapidly and require less overhead than WoLF-PHC's definition [8, 26].

3. **Winning and learning step rates.** These are the step sizes discussed above, δ_w, and δ_l. These values determine the learning rate and affect the rate at which an agent will converge on an optimal solution. In [26], the authors suggest a win-to-learn ratio of 1:3 for good performance, however, since DA*rep* is a different problem, many different ratios were experimented with in order to determine the optimal parameters for WoLFAntDA(see section 4.3.

WoLF has been adapted to help solve many problems such as the Traveling Salesman Problem [26] and stochastic matrix games [8, 11, 12, 27]. Section 3.2 describes how the Win or Learn Fast algorithm has been combined with AntDA in order to solve the Quality-Sensitive DA Replication Problem.

This chapter described the Quality-Sensitive DA Replication Problem as well as other assignment problems related to it. The ACO and WoLF algorithms were covered as well and how they have been adapted to fit other optimization problems.

III. Methodology

This chapter begins with an overview of how the Ant Colony Optimization and Win or Learn Fast Algorithms were adapted in order to be implemented on the Quality-Sensitive DA Replication Problem. Next, there is an introduction to other solution methods used for performance comparison with the proposed algorithms: AntDA, WoLFAntDA, and PD-WoLFAntDA. This chapter concludes with an explanation of the Server Filling heuristic which was combined with the Ant Colony Optimization algorithm to enhance the performance of AntDA.

*3.1 AntDA: An ACO Algorithm for DA*rep

The first proposed algorithm, AntDA, is the adaptation of ACO to fit the Quality-Sensitive DA Replication Problem. This section covers the basic behavior, transition rules, and the rules for depositing pheromone on edges.

3.1.1 Basic Behavior. In AntDA, ants operate on a bipartite graph representing an instance of DA*rep* (Fig. 6). The graph, $G = (V, E)$, consists of a set of vertices, V, and edges connecting vertices, E. The vertices are divided into two groups, DQ and S, such that $V = DQ \cup S$ and $DQ \cap S = \emptyset$. Each vertex in DQ represents a $\langle d, q \rangle$ pair (a quality q of DA d).

The vertices in S represent the servers. Each $dq \in DQ$ is connected to every $s \in S$ by a directed edge (dq, s). Similarly each $s \in S$ is connected to every $dq \in DQ$ by a directed edge (s, dq). Even though each edge (dq, s) has a reverse edge (s, dq), undirected edges are not used since pheromone is interpreted differently on edges of type (dq, s) versus edges of type (s, dq).

Ants construct solutions by moving back and forth between vertices in DQ ($\langle d, q \rangle$ pairs) and vertices in S (servers) creating a replica and assigning dq request load or adjusting the service quality of an existing replica on the server chosen for assignment.

Ants work independently (maintain their own solution spaces) but share the same graph. An ant works on a solution until either server capacity is exhausted or all request

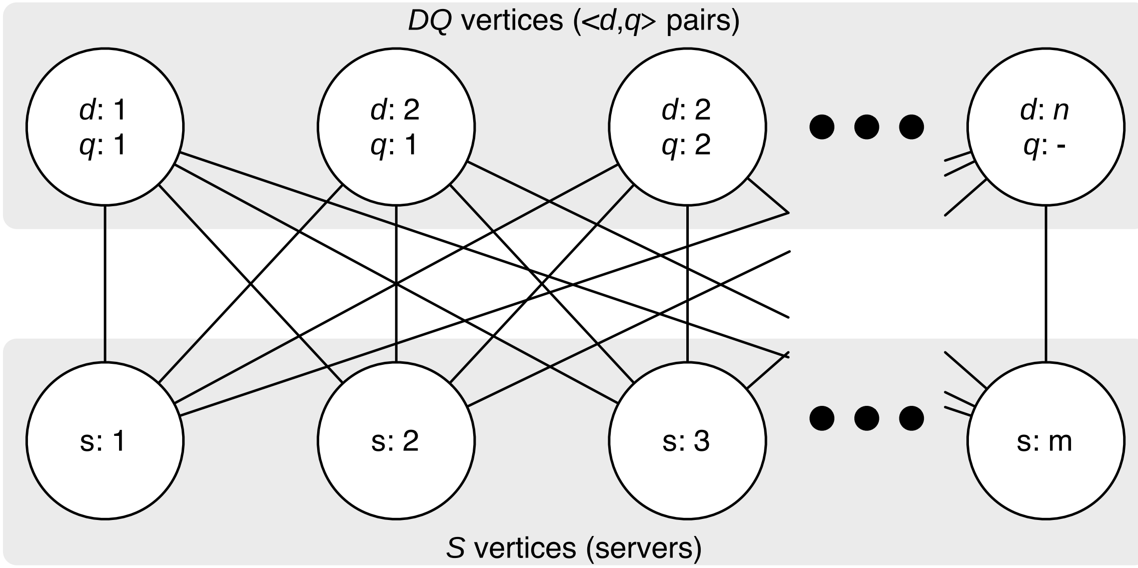

Figure 6: The bipartite problem graph used by AntDA. Although all $\langle d, q \rangle$ pairs are connected to servers via directional edges and vice-versa, single non-directional edges are shown here for simplicity.

load has been assigned to the servers. Once all ants have solved, they deposit pheromone on the shared graph, pheromone evaporation takes place, and then the next time step begins. Ants are placed at a random server vertex at the beginning of each time step. The algorithm for AntDA is shown in algorithm 1.

3.1.2 Moving From Servers to $\langle d, q \rangle$ pairs. An ant at vertex s must decide which $\langle d, q \rangle$ pair should be assigned next (Algorithm 1, step 14). Let DQ_s^k be the set of dq vertices which are still capable of being assigned to servers. A $\langle d, q \rangle$ pair can be assigned if:

1. it has some amount of unassigned request load ($rem(rl_{d,q}) > 0$), and
2. There exists a server s such that the net change in update load on s because of placing a replica of DA d at quality q on s is less than the remaining capacity on s: $rem(C_s) >$ *net change in* ul_s *because of hosting a replica of the* $\langle d, q \rangle$ *pair.*

Algorithm 1 The AntDA Algorithm

Initialize parameter values
for each edge $(dq, s) \in$ graph G **do**
  $\tau_{(dq,s)} = \tau_0$
end for
for each edge $(s, dq) \in$ graph G **do**
  $\tau_{(s,dq)} = \tau_0$
end for
for each time step t **do**
  Distribute graph G to all ants
  for each ant k **do**
    $T^k = \emptyset$
    Randomly select a starting server $s \in S$
    while $DQ_s^k \neq \emptyset$ and $S_{dq}^k \neq \emptyset$ **do**
      Select $dq \in DQ_s^k$ according to equation 12
      Select $s \in S_{dq}^k$ according to equation 14
      Assign dq to s.
      Adjust server capacity to reflect assignment.
      Adjust dq remaining load.
      Update DQ_s^k and S_{dq}^k
      Invoke the server filling algorithm - Section 3.3
    end while
  end for
  //Now all ants have built tours for time step t
  for each ant k with one of the top m solutions **do**
    Update pheromone using the rules in Section 3.1.4
  end for
end for

If $DQ_s^k = \emptyset$, the algorithm terminates. Otherwise, the probability that ant k selects edge (s, dq) is given by:

$$p_{s,dq}^k(t) = \begin{cases} \dfrac{[\tau_{s,dq}(t)]^\alpha \cdot [\eta_{s,dq}]^\beta}{\sum\limits_{dq' \in DQ_s^k} [\tau_{s,dq'}(t)]^\alpha \cdot [\eta_{s,dq'}]^\beta}, & \text{when } dq \in DQ_s^k \\ 0, & \text{when } dq \notin DQ_s^k. \end{cases} \tag{12}$$

where $\tau_{s,dq}(t)$ is the pheromone concentration on (s, dq). The scaling parameters α and β again control the relative importance of pheromone and heuristic desirability. Also, $\eta_{s,dq}$ is

the heuristic desirability of selecting (s, dq) and is given by:

$$\eta_{s,dq} = \frac{ul_{dq} \cdot rem(rl_{dq})}{\sum\limits_{dq' \in DQ_s^k} ul_{dq'} \cdot rem(rl_{dq'})}. \tag{13}$$

Dividing $ul_{d,q} \cdot rem(rl_{d,q})$ by a server's remaining capacity, $rem(C_s)$, estimates the update burden incurred by creating replicas on servers of size $rem(C_s)$. Eq. (13) is an appropriate heuristic since it prefers $\langle d, q \rangle$ pairs most likely to produce high update burdens (no matter which servers are used). Note that $\eta_{s,dq}$ values change as the ant constructs its solution.

After making its selection, the ant traverses the edge to the selected $\langle d, q \rangle$ pair and then must choose a new server.

3.1.3 Transitioning From $\langle d, q \rangle$ pairs to Servers. When at vertex dq, ant k must find a server to which the $\langle d, q \rangle$ pair represented by dq will create a replica and assign load (Algorithm 1, step 15). Let S_{dq}^k be the set of servers (vertices) upon which request load of the $\langle d, q \rangle$ pair represented by vertex dq can be assigned. Let $rem(C_s)$ represent the unused (remaining) capacity of server s. Server s is available for assignment if the net change in update load on s caused by its hosting DA d at quality q is less than $rem(C_s)$ (i.e., s will be able to handle request load for the $\langle d, q \rangle$ pair). This is the **server hosting condition**.

The probability that ant k selects edge (dq, s) is

$$p_{dq,s}^k(t) = \begin{cases} \dfrac{[\tau_{dq,s}(t)]^\alpha \cdot [\eta_{dq,s}]^\beta}{\sum\limits_{s' \in S_{dq}^k} [\tau_{dq,s'}(t)]^\alpha \cdot [\eta_{dq,s'}]^\beta}, & \text{when } s \in S_{dq}^k \\ 0, & \text{when } s \notin S_{dq}^k. \end{cases} \tag{14}$$

$\tau_{dq,s}(t)$ is the pheromone concentration on (dq, s) at time step t. Parameters α and β are constants governing the relative importance of pheromone to the heuristic desirability, $\eta_{dq,s}$, of traveling along edge (dq, s):

$$\eta_{dq,s} = \frac{rem(C_s)}{\sum\limits_{s' \in S_{dq}^k} rem(C_{s'})} \tag{15}$$

where $rem(rl_{d,q})$ is the amount of request load for $\langle d, q \rangle$ pair yet to be assigned to a server. The heuristic is based on the idea that greedily selecting the largest server should reduce the number of replicas created and, thus, update burden produced. Note that $\eta_{dq,s}$ values change as servers are assigned. The heuristic in Equation (15) mirrors the greedy selection criteria of the greedy algorithm (see Section 3.4) in the way that it favors the selection of the server with the most remaining capacity.

After selecting edge (dq, s) the ant moves from vertex dq to vertex s. Once at s the ant creates a replica for the $\langle d, q \rangle$ pair and assigns as much remaining request load of the $\langle d, q \rangle$ pair, $rem(rl_{dq})$, to s as possible. If a replica of DA d already exists on s, then the ant adjusts the quality level of the replica if needed (increases the update load of the replica). The server's remaining capacity, $rem(C_s)$, is decreased based on the amount of update load and request load assigned.

After creating a replica of DA d at quality level q on server s, the ant can attempt to invoke the **Server-Filling (SF)** heuristic (explained in section 3.3). In cases where a replica of d does not use all the capacity on its host server s, the SF heuristic looks to assign request load of other qualities of d to s. After making its selection, the ant traverses the edge to the selected server node and then transitions back to a $\langle d, q \rangle$ pair (Section 3.1.2).

3.1.4 Pheromone Update Rule. When each ant has constructed a solution to DA*rep* it is time to deposit pheromone on the shared graph (Algorithm 1, step 25). By finding a solution, an ant has essentially assigned values for the $x_{s,d,q}$ and $\lambda_{s,d,q}$ variables described in the informal version of the problem shown in Fig. 4.

Recall that the DA*rep* problem's goal is to minimize the amount of update load, UL, expended in a solution that assigns all request load. In other words, DA*rep* seeks to minimize the following equation:

$$UL = \sum_{s \in S} \sum_{d \in D} \sum_{q \in Q_d} ul_{d,q} \cdot x_{s,d,q}. \tag{16}$$

Quite naturally, the minimization function of the formal problem (Eq. 16) can be used to rate the solutions found by the ants. The number of ants allowed to update edges and exactly how much pheromone each updating ant deposits is a tunable parameter subject to experimentation. The pheromone deposit scheme which worked best for AntDA is explained in section 4.2.

Since better solutions have lower update burdens, the amount of pheromone deposited by ants is inversely proportional to a solution's update burden. However, low update burdens are not always better – since some ants' solutions may be infeasible (i.e., they do not assign all request load). Differentiating between feasible and infeasible solutions when deciding how much pheromone to deposit on the edges used in an ants solution is easily handled. Let $UB_k(t)$ be the update burden of ant k's solution after time step t as computed by (3). Then adjust $UB_k(t)$ to account for infeasible assignments as follows:

$$UB'_k(t) = \frac{UB_k(t)}{\left(\frac{RL_k(t)}{RL}\right)^{\omega}} \tag{17}$$

where $RL_k(t)$ is the amount of request load assigned by ant k in time step t and ω is a constant that determines the magnitude of the penalty paid for not assigning all request load. Eq. (17) increases the update load of an infeasible assignment based on how much request load was satisfied raised by ω. Thus, infeasible assignments cannot compete with feasible ones.

Once $UB'_k(t)$ has been determined, it is used to calculate the amount of new pheromone ant k will deposit. The ants with the m best solutions are allowed to deposit pheromone after each time step. More specifically, if edge e was used in the ith best solution and $i \leq m$, then the amount of pheromone deposited on e by the ant that produced the ith best solution is

$$\Delta\tau_e(t) = \frac{\gamma}{UB'_i(t)} \tag{18}$$

where γ is a constant. For AntDA, γ was set to 1 during experimentation and was found to have little, if any, impact on performance. If an edge e was not used by ant i, then $\Delta_e^i(t) = 0$.

Let $\Delta\tau_e(t) = \sum_{i=1}^{m} \Delta_e^i(t)$ be the amount of new pheromone to be deposited on edge e because of the m solutions chosen. The amount of pheromone on the edges in graph G is then updated as is typically done in ACO [9]:

$$\tau_e(t+1) \leftarrow (1-\rho) \cdot \tau_e(t) + \rho \cdot \Delta\tau_e(t) \tag{19}$$

Once implemented, AntDA's parameters were tuned, the server filling heuristic added, and simulations run. Section 4.4 describes the performance of AntDA and how it compared to other search algorithms in solving the Quality-Sensitive DA Replication Problem.

*3.2 WoLFAntDA: A Reinforcement Learning ACO algorithm for DA*rep

3.2.1 Motivation: Why WoLF? Although the Ant Colony Optimization algorithm has very good search capability in optimization problems, it still has some drawbacks such as stagnation, computing time, and premature convergence. Stagnation and premature convergence can be limited by tuning of parameters for each individual problem. However, computing time is due to random decision making and problem/graph size and can only be slightly reduced by parameter tuning. Thus, ACO is not, at present, an effective method for some problems.

In AntDA, tuning the parameters of the ACO algorithm allowed for better solutions and quicker convergence (see Section 4.7. However, there is still room for improvement. In [26], the authors' combined a variable-step policy hill-climbing algorithm called Win or Learn Fast with an ACO algorithm for solving the Traveling Salesman problem. They found that the addition of this learning algorithm provided faster convergence to optimal solutions. Therefore, in an attempt to achieve the goals of faster convergence and better solution values for larger problems, WoLFAntDA and PD-WoLFAntDA combine the AntDA algorithm

with two versions Win or Learn Fast (WoLF-PHC [12] and PDWoLF-PHC [8] respectively). These algorithms were explained in section 2.5.2.

3.2.2 How the Win or Learn Fast algorithm was modified to work with AntDA.

As described in section 2.5.2, there are three characteristics that must be modified for each specific problem in which it is applied. The three options manipulated to use the Win or Learn Fast algorithm (introduced in section 2.5.2) in both WoLFAntDA and PD-WoLFAntDA are:

1. **An estimation policy.** In AntDA, each ant traverses the problem graph and constructs a solution. Following, the ants with the top m solutions deposit pheromone on the edges used to construct their solutions. Therefore, the pheromone concentration on an edge is the best estimate available as to which edges should be used to construct the optimal solution. Hence, the estimation policy used in WoLFAntDA and PD-WoLFAntDA is edge pheromone.

2. **A rule for determining winning or losing**. There have been two suggested rules for determining winning or losing in the Win or Learn Fast algorithm. Both of these methods have been described in section 2.5.2. Although it has been shown that the PDWoLF-PHC definition of winning and losing requires less computation and memory overhead and converges more rapidly in other problems than the WoLF-PHC definition [26], neither one has been applied to DA*rep*. Therefore, AntDA has been implemented with both rules (adapted to ACO) to determine which allows for better solutions and convergence rates for this problem. WoLFAntDA was implemented with the WoLF-PHC definition and PD-WoLFAntDA with the PDWoLF-PHC definition. The implementations are described in the following sections and results are shown in section 4.7.

3. **Winning and learning step rates.** After testing multiple values, in the same manner as in parameter selection, the step rates to be used in WoLFAntDA and PD-WoLFAntDA are δ_w set to 0.005 and δ_l set to 0.030. These effect how the policy values for an edge are updated (more fully explained in 3.2.7).

Algorithm 2 The WoLFAntDA and PD-WoLFAntDA Algorithm

Initialize parameter values
for each edge $(dq, s) \in$ graph G **do**
  $\tau_{(dq,s)} = \tau_0$
end for
for each edge $(s, dq) \in$ graph G **do**
  $\tau_{(s,dq)} = \tau_0$
end for
for each time step t **do**
  Distribute graph G to all ants
  for each ant k **do**
    $T^k = \emptyset$
    Randomly select a starting server $s \in S$
    while $DQ_s^k \neq \emptyset$ and $S_{dq}^k \neq \emptyset$ **do**
      Select $dq \in DQ_s^k$ according to equation 20
      Select $s \in S_{dq}^k$ according to equation 21
      Assign dq to s.
      Adjust server capacity to reflect assignment.
      Adjust dq remaining load.
      Update $dq \in DQ_s^k$ and $s \in S_{dq}^k$
      Invoke the server filling algorithm - Section 3.3
    end while
  end for
  //Now all ants have built tours for time step t
  for each ant k with one of the top m solutions **do**
    Update pheromone using the rules in section 3.2.6
  end for
  for each ant k with one of the top p solutions **do**
    Update policy using the rules in section 3.2.7
  end for
end for

3.2.3 Basic Behavior. In WoLFAntDA and PD-WoLFAntDA, ants still operate on a bipartite graph representing an instance of DA*rep* (Fig. 6) and make decisions much as they do in AntDA. The main difference is that decision-making is now sensitive to edge pheromone, the heuristic desirability of the edge, and edge policy values. Each edge (whether server to $\langle d, q \rangle$ pair or $\langle d, q \rangle$ pair to server) maintains each of these values and each are updated using the methods described in the following sections. The algorithm for WoLFAntDA and PD-WoLFAntDA is shown in algorithm 2.

3.2.4 Moving From Servers to $\langle d,q \rangle$ pairs. An ant at vertex s must decide which $\langle d,q \rangle$ pair should be assigned next (Algorithm 2, step 14). The selection of the $\langle d,q \rangle$ pair to be assigned is subject to the same constraints as in AntDA(see section 3.1.2) and the decision making process has also been adjusted slightly to allow policies to affect $\langle d,q \rangle$ pair selection.

The probability that ant k selects edge (s, dq) is given by:

$$p^k_{s,dq}(t) = \begin{cases} \frac{[\tau_{s,dq}(t)]^\alpha \cdot [\pi_{s,dq}(t) \cdot \eta_{s,dq}]^\beta}{\sum\limits_{dq' \in DQ^k_s} [\tau_{s,dq'}(t)]^\alpha \cdot [\pi_{s,dq'}(t) \cdot \eta_{s,dq'}]^\beta}, & \text{when } dq \in DQ^k_s \\ 0, & \text{when } dq \notin DQ^k_s. \end{cases} \tag{20}$$

where $\tau_{s,dq}(t)$ is the pheromone concentration on (s, dq) and $\pi_{s,dq}(t)$ is the policy value on (s, dq) at time step t. The scaling parameters α and β again control the relative importance of pheromone and policy/heuristic desirability. Once again, the heuristic desirability for the transition from a server to $\langle d,q \rangle$ pair is calculated the same in WoLFAntDA and PD-WoLFAntDA as in AntDA and is shown in Eq. 13.

Equation 20 is identical to Eq. 12 except for the addition of the policy term, π. The introduction of this term allows the ants to be sensitive to pheromone, policy, and heuristic values when selecting a $\langle d,q \rangle$ pair.

After making its selection, the ant traverses the edge to the selected $\langle d,q \rangle$ pair and then must choose a server node.

3.2.5 Transitioning From $\langle d,q \rangle$ pairs to Servers. When at vertex dq, ant k must select a server to which the $\langle d,q \rangle$ pair represented by dq will create a replica and assign load (Algorithm 2, step 15). This selection of the server follows the AntDA method but with slight changes that allow policies to affect server selection.

The probability that ant k selects edge (dq, s) is

$$p^k_{dq,s}(t) = \begin{cases} \frac{[\tau_{dq,s}(t)]^\alpha \cdot [\pi_{dq,s}(t) \cdot \eta_{dq,s}]^\beta}{\sum\limits_{s' \in S^k_{dq}} [\tau_{dq,s'}(t)]^\alpha \cdot [\pi_{dq,s'}(t) \cdot \eta_{dq,s'}]^\beta}, & \text{when } s \in S^k_{dq} \\ 0, & \text{when } s \notin S^k_{dq}. \end{cases} \tag{21}$$

where $\tau_{dq,s}(t)$ is the pheromone concentration on (dq, s) at time step t and $\pi_{dq,s}(t)$ is the policy value on (dq, s) at time step t. α and β are constants governing the relative importance of pheromone to the policy/heuristic desirability of traveling along edge (dq, s). The heuristic desirability is calculated the same in WoLFAntDA and PD-WoLFAntDA as in AntDA and is shown in Eq. 15.

Equation 21 is identical to Eq. 14 except for the addition of the policy term, π. The introduction of this term makes the ants to be sensitive to pheromone, policy, and heuristic values when selecting a server. Early in the algorithm, when pheromone and policies are fairly neutral, ants are guided by the heuristics, η. However, as pheromone and policies become more differentiated, ant behavior becomes more dependent on them [26].

After selecting edge (dq, s) the ant moves from vertex dq to vertex s and updates its graph in the same manner as AntDA. It then transitions back to a dq node (Section 3.2.4

3.2.6 Pheromone Update Rule. When each ant has constructed a solution to DA*rep* it is time to deposit pheromone on the shared graph (Algorithm 2, step 25). For WoLFAntDA and PD-WoLFAntDA, this process is done in the exact manner as AntDA which is described in section 3.1.4.

3.2.7 Policy Updates. After all ants have constructed solutions and pheromone has been deposited on the shared graph, it is time to update policy values for the edges (Algorithm 2, step 28). To regulate the policy update for these edges, the following equations were used. This section has been adapted from [8,11] where a similar approach was used to merge WoLF with ACS-TSP (an ACO algorithm) to solve a Traveling Salesman Problem in [26]. The variables used for policy update depend on what kind of edge is being updated.

3.2.7.1 Server to $\langle d, q \rangle$ pair. The first step (Equations 22 and 23) is to determine whether the particular edge is *winning* or *losing*. Equation 22 is the WoLF-PHC definition [12] for *winning* and *losing* while equation 23 is the PDWoLF-PHC definition [8]. In section 2.5.2, these two different rules were discussed in depth. The equations remain similar to those of [12] and [8] except for a minor notation change. Both of these equations originally used the idea of state/action pairs (explained in section 2.5.2), but this concept has been adapted to fit the pheromone and graph nature of AntDA. In WoLFAntDA and PD-WoLFAntDA, for an ant traversing from a server to $\langle d, q \rangle$ pair, the state is the server, s,the ant has currently chosen, and the action is one of the $\langle d, q \rangle$ pairs, dq, capable of being hosted by the server s. These equations reveal the learning rate, δ, for the given state/action pair. The learning rate for WoLFAntDA is given by:

$$\delta = \begin{cases} \delta_w, \text{ if } \sum\limits_{dq' \in DQ_s} \pi_{(s,dq')} \tau_{(s,dq')} > \sum\limits_{dq' \in DQ_s} \overline{\pi}_{(s,dq')} \tau_{(s,dq')} \\ \delta_l, \text{ otherwise} \end{cases} \tag{22}$$

While the learning rate for PD-WoLFAntDA is given by:

$$\delta = \begin{cases} \delta_w, \text{ if } \Delta\pi_{(s,dq')}(t) \cdot \Delta\pi^2_{(s,dq')}(t) < 0 \\ \delta_l, \text{ otherwise} \end{cases} \tag{23}$$

where

$$\Delta\pi_{(s,dq')}(t) = \pi_{(s,dq')}(t) - \pi_{(s,dq')}(t-1) \tag{24}$$

and

$$\Delta\pi^2_{(s,dq')}(t) = \Delta\pi_{(s,dq')}(t) - \Delta\pi_{(s,dq')}(t-1) \tag{25}$$

Once an edge's learning rate is discovered, a policy change value must be calculated, which is given by equations 26 and 27.

$$\delta_{(s,dq)} = \min\left\{\pi_{(s,dq)}, \frac{\delta}{|DQ|-1}\right\} \tag{26}$$

$$\Delta_{(s,dq)} = \begin{cases} -\delta_{(s,dq))}, dq \neq \arg\max_{l \in DQ} \tau_{(s,l)} \\ \sum_{l \in S} \delta_{(l,dq))}, \text{ otherwise} \end{cases} \tag{27}$$

Once the change in policy is determined, it is used to calculate the new policy value for the edge by adding the change in policy to the current policy value.

$$\pi_{(s,dq)}(t+1) = \pi_{(s,dq)}(t) + \Delta_{(s,dq)} \tag{28}$$

The ants with the p best solutions were allowed to update policy on the edges. Each of these ants computes $\Delta_{(s,dq)}$ and then the sum of all of them is used in equation 28. p is a tunable parameter subject to experimentation. The parameter values that worked best for WoLFAntDA and PD-WoLFAntDA are explained in section 4.3.

3.2.7.2 $\langle d, q\rangle$ pair s to Servers. Once policy has been updated on the server to $\langle d, q\rangle$ pair edges, policy is then updated on the $\langle d, q\rangle$ pair to server edges. This is accomplished in the same manner as section 3.2.7.1 except that instead of using the edges going from servers to $\langle d, q\rangle$ pairs ($\pi_{(s,dq)}$), the edges traversing from $\langle d, q\rangle$ pairs to servers ($\pi_{(dq,s)}$) are used.

3.3 The Server-Filling Replica Creation Heuristic

In the AntDA, WoLFAntDA, and PD-WoLFAntDA algorithms described previously in this chapter, ants simply create replicas or adjust the update loads of existing replicas (when the replica is already hosting a lower quality replica of application d) as they make assignments. However, in cases where a replica of DA d does not use all the capacity on its host server s, it may be possible to assign request load of other qualities of d to s. If

additional request load can be assigned to s, then update burden on s can be further utilized and, in turn, system-wide update burden, UB, can be kept low.

AntDA, WoLFAntDA, and PD-WoLFAntDA do this using the Server Filling heuristic (SF). It can be invoked when the following two conditions are met.

1. SF first tries to avoid the creation of extra replicas of d by finding other qualities of d that completely fit on s. More specifically, SF looks for another quality $r \in Q_d$ such that all of $rem(rl_{d,r})$ can be assigned to s. Note that update load differences have to be accounted for since it may be that $r > q$ and hence $ul_{d,r} > ul_{d,q}$. SF assigns the highest r found, repeating with additional qualities of d if possible. Let y be the highest quality of d assigned to s at the end of this step.

2. If s still has spare capacity after step 1, SF looks for the highest quality y of d such that $u < y$ and assigns as much request load of quality u as possible to the replica.

The SF heuristic is an optional, but beneficial, part of AntDA, WoLFAntDA, and PD-WoLFAntDA; in experiments SF reduced update burden by over 4% on average (see Section 4.5).

3.4 Other Solution Methods Used for Comparison

To show the worth of AntDA, WoLFAntDA, and PD-WoLFAntDA, they must be shown to perform better than other solution methods that have historically been used for assignment optimization problems. In the next chapter, these three algorithms' results are compared against three algorithms adapted to fit the Quality-Sensitive DA Replication Problem. Results for performance comparisons were obtained by using a random assignment algorithm, Random, a greedy algorithm, Greedy, and the LINGO Integer Linear Programming (ILP) solver [50].

Random picks a $\langle d, q \rangle$ pair with non-zero remaining request load at random and assigns it to a random server capable of hosting it. All selections are made using a uniform distribution. Random reports the best solution found out of 1000 trials.

The **Greedy** algorithm [53] makes assignments by choosing the $\langle d, q \rangle$ pair with the highest predicted update burden. It does this by the following algorithm:

Algorithm 3 The Greedy Algorithm for DA*rep*

1: Sort the set of capacitated servers by capacity and store the results in a data structure $\mathcal{S}$.
2: Set $rem(rl_{d,q}) = rl_{d,q}$ for each $\langle d, q \rangle$ pair.
3: Let $C_{\max}$ denote the capacity of the server, $s_{\max}$, with the most remaining capacity in $\mathcal{S}$. Choose the $\langle d, q \rangle$ pair to be assigned to $s_{\max}$
4: **if** creating a replica of the chosen $\langle d, q \rangle$ pair on $s_{\max}$ means that $s_{\max}$ has no room left over for handling requests ($C_{\max} \leq ul_{d,q}$) **then**
5:     remove $s_{\max}$ from $\mathcal{S}$ and go to step 19.
6: **end if**
7: For the $\langle d, q \rangle$ pair selected in Step 2, decide the replica's quality $repQ$ ($repQ \geq q$)
8: Then, decide the amount of request load for any additional qualities of d, $r \in Q_d$, to be carried by the replica using either the Server Filling replica creation policy (Section 3.3).
9: Record $repQ$ and the amount of request load of each $r \in Q_d$ assigned to $s_{\max}$.
10: Decrement $rem(rl_{d,r})$ for each $r \in Q_d$ by the amount assigned to $s_{\max}$.
11: Decrement C_{max} by the replica's update load and the sum of the request loads assigned.
12: **if** $rem(rl_{d,q}) = 0$ for all $\langle d, q \rangle$ pairs **then**
13:     STOP with a complete solution.
14: **end if**
15: **if** $C_{\max} = 0$ **then**
16:     remove $s_{\max}$ from $\mathcal{S}$.
17: **end if**
18: Resort $\mathcal{S}$ if needed.
19: **if** $\mathcal{S} = \emptyset$ **then**
20:     STOP with a partial solution.
21: **else**
22:     go to step 3.
23: **end if**

Greedy only needed to be run once for its best solution to be found.

LINGO ILP Solver solves DA*rep* using the ILP formulation in section 2.2.4. Although ILP solvers such as LINGO are the only known method besides complete enumeration that can find guaranteed optimal solutions, execution times can be prohibitive. Therefore, LINGO was only used on the small test cases and allotted four hours to work on the DA*rep* problems. This was sufficient time for LINGO to product feasible, but not optimal, solutions and provides a notion of DA*rep*'s complexity.

All three of the assignment algorithms presented above are static. For more information on some dynamic assignment algorithms adapted for DA*rep*, as well as more detailed explanations of these static algorithms, see [53].

Chapter IV presents the results of experiments that compare AntDA with these solution methods, reveal the importance of the Server-Filling heuristic, and the importance of pheromone and heuristics on ants traversing the bipartite graph. It also demonstrates the effects of the Win or Learn Fast algorithm combined with AntDA compared with AntDA alone.

IV. Results and Analysis

This chapter evaluates the performance of the proposed AntDA, WoLFAntDA, and PD-WoLFAntDA algorithms and is divided into six main sections. Section 4.1 discusses the configuration of the experiments used for analysis. The second and third sections describe how parameter values are chosen and which values are used for both AntDA and WoLFAntDA. Section 4.4 demonstrates the performance of AntDA as compared with other solution methods. Sections 4.5 and 4.6 show the effect of the Server Filling optimization heuristic and of limiting the number of ants allowed to deposit pheromone has on the three proposed algorithms. The chapter concludes with a performance analysis of WoLFAntDA and PD-WoLFAntDA versus AntDA.

4.1 Explanation of Test Cases

Each experiment involves a hypothetical ASP with a variety of server capacities and customer DAs. The DAs are designed to subject the algorithms to extremes that might be found in a real world environment. The test cases were derived from [53].

In each experiment, DAs have the same number of service quality levels (either 1, 2, or 3) and have a particular update load (UL) pattern and request load (RL) pattern. Table 2 describes the parameters used in constructing ASPs for the experiments.

The UL pattern determines the update load values that the freshness quality levels of the DA can assume. There are two patterns: low and high. Update loads for DAs always increase with quality level. The low UL pattern ensures that all qualities of all the DAs can fit on any of the ASP's servers. However, update loads in the high UL pattern are boosted so that some servers will not be able to host high-quality replicas of some DAs.

The RL pattern determines how user request loads change with quality level for the DAs an ASP is hosting. There are two patterns: increasing and decreasing. For the decreasing RL pattern, request loads are large for low quality levels and decrease as quality levels rise. For the increasing RL pattern, request loads start small and increase with quality level.

Parameter	Parameter Option	Option Description
Number of Qualities Per DA	1	All the DAs hosted by the ASP have 1 service quality level.
	2	All the DAs in an ASP have 2 service quality levels.
	3	All the DAs in an ASP have 3 service quality levels.
Update Load (UL) Pattern	`low`	Any server can host any DA.
	`high`	The maximum update load can be above the capacity limit of the smallest servers.
Request Load (RL) Pattern	`decreasing`	Request load decreases as a DB's quality levels increase. Since higher service qualities require more maintenance and probably account for declining percentages of demand, this pattern is most likely predominant in the real world.
	`increasing`	`decreasing`'s opposite – the higher the service quality, the higher the request load.

Table 2: Parameters Used in Constructing ASPs for the Static Experiments.

# DAs Per ASP	# Quals Per DA	UL Pattern and Value Ranges For Each DA Quality Level				RL Pattern and Value Ranges For Each DA Quality Level			
		Option	1	2	3	Option	1	2	3
5	1	`low`	1-5	-	-	`decr`	400-600	-	-
5	1	`high`	5-35	-	-	`decr`	400-600	-	-
5	2	`low`	1-7	9-15	-	`decr`	300-400	100-200	-
5	2	`high`	5-18	22-35	-	`incr`	100-200	300-400	-
5	3	`low`	1-4	6-9	11-15	`decr`	233-300	133-200	34-100
5	3	`high`	5-14	16-25	27-36	`incr`	34-100	133-200	233-300
10	3	`low`	1-4	6-9	11-15	`decr`	233-300	133-200	34-100
10	3	`high`	5-14	16-25	27-36	`incr`	34-100	133-200	233-300
20	3	`low`	1-4	6-9	11-15	`decr`	233-300	133-200	34-100

Table 3: How ASPs Were Constructed for the Static Experiments. This table shows how the parameters of Table 2 were combined to form ASPs for the static experiments. Value ranges for update loads and request loads are listed.

Each experiment is run on a Intel Xeon CPU, 3.20 GHz processor with 3.75 GB of RAM. AntDA, WoLFAntDA, and PD-WoLFAntDA were all written in Java and run in Netbeans 4.0 development environment.

Table 3 shows how these parameters were combined to produce the test cases for the static experiments. For each combination of parameters (UL pattern, RL pattern, and # of qualities/DA) five, ten, or twenty ASPs were randomly created. For example, the five 2-quality/low/decreasing ASPs (third row of data in Table 3) shows that DA update loads in these type of ASPs range from 1-7 for quality 1 and 9-15 for quality 2, while

request loads range from 300-400 for quality 1 and 100-200 for quality 2. Five test cases were also generated for a 20 DA, 3 quality, increasing RL pattern, high UL pattern but the hardware the tests were conducted on was insufficient to compute these cases due to its high complexity levels.

Once an ASP's DAs are assigned, the ASP's servers were determined by growing a candidate set of servers. Initially empty, servers are added to the candidate set in groups of five. The servers in each group have the following load capacities: 25, 50, 75, 100, 125. This distribution of server capacities is intended to model an ASP with a variety of servers. Groups are added to the candidate set until the Greedy algorithm, using the Server Filling policy (see section 4.5), produced feasible assignments. These test cases were then solved by Random, Greedy, and the LINGO ILP Solver as well as AntDA and WoLFAntDA. The results are shown and discussed in sections 4.4 and 4.7.

4.2 Parameter Selection for AntDA

The main task after implementing AntDA is to find the best parameter values in order to optimize them for solving the Quality-Sensitive DA Replication Problem. The parameter values are determined through trial and error, testing many different values for each parameter subjected to 50 trials of 400 time steps each. Since there are many parameters, trying every possible combination of them would be infeasible. Therefore, to determine parameter values, one parameter is chosen as the variable to be tested and all others are held constant. Table 4 shows each parameter and the values examined/tested for each. The numbers in **bold** are the values for parameters while they are being held constant. Each test case was run using the same problem instance to maintain consistency. The problem instance is one of the five constructed from line ten of Table 3 (# nodes = 70) because it was the hardest problem available that could be solved in a manageable amount of time by AntDA and LINGO (though LINGO only produced feasible but not optimal solutions). After going through this process, the best value for each parameter was used in combination with each other to verify that performance did not degrade (it didn't). The best values

Parameter	Parameter Values Tested
α	$0, \mathbf{1}, 3, 5, 7, 8, 10$
β	$0, 1, 3, 5, 7, \mathbf{8}, 10$
ρ	$0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, \mathbf{0.8}, 0.9$
ω	$1, 2, 3, \mathbf{4}, 5, 6, 7, 8, 9$
γ	$0.01, \mathbf{0.1}, 1, 10, 100$
Number of Ants	$35, 42, 49, 56, 63, \mathbf{70}, 77, 84, 91, 98, 105$
τ_0	$0.0000001, 0.000001, 0.00001, 0.001, \mathbf{0.1}, 1.1, 5, 10, 100, 500, 1000$
m	$1, 3, 7, 14, 21, 28, 35, 42, 49, 56, 63, \mathbf{70}$

Table 4: Parameter and condition values tested for AntDA experiments.

Parameter	Parameter Value	Parameter Description
α	1	Pheromone weighting.
β	8	Heuristic weighting.
ρ	0.8	New to old pheromone ratio.
ω	4	Non-feasible solution penalty constant.
γ	1	Pheromone change constant.
Number of Ants	$\lvert DQ\rvert + \lvert S\rvert$	The number of ants.
τ_0	0.1	Initial edge pheromone.
m	$\lfloor \#\text{ Ants} \cdot 0.1 \rfloor$	The top m ants are allowed to deposit pheromone.

Table 5: Parameter and condition values for AntDA experiments.

identified by this selection process are shown in table 5 and are used throughout the AntDA experiments presented hereafter unless otherwise stated.

For the most part, AntDA was fairly insensitive to a change in the values shown in Table 5. However, the one parameter that had a major impact was the number, m, of ants that deposit pheromone at the end of each time step (Sections 3.1.4). For AntDA, setting m to be the top 10% of the number of ants cut the convergence rate by as much as 4.5 times compared to allowing all ants to deposit pheromone while also reducing update burdens. The impact of setting m to the top 10% of the number of ants is presented in section 4.6.

4.3 Parameter Selection for WoLFAntDA and PD-WoLFAntDA

After implementing WoLFAntDA and PD-WoLFAntDA the parameters needed to be tuned for solving the Quality-Sensitive DA Replication Problem. The parameter values

Parameter	Parameter Values Tested
α	$0, \mathbf{1}, 3, 5, 7, 8, 10$
β	$0, 1, 3, 5, 7, \mathbf{8}, 10$
m	$1, 3, \mathbf{7}, 14, 21, 28, 35, 42, 49, 56, 63, 70$
p	$1, 3, \mathbf{7}, 14, 21, 28, 35$
δ_l	$0.005, 0.01, \mathbf{0.015}, 0.02, 0.025, 0.03, 0.035$
δ_w	$\mathbf{0.005}, 0.01, 0.015, 0.02, 0.025, 0.03, 0.035$

Table 6: Parameter and condition values tested for the WoLFAntDA and PD-WoLFAntDA experiments.

Parameter	Parameter Value	Parameter Description
α	1	Pheromone weighting.
β	8	Heuristic weighting.
ρ	0.8	New to old pheromone ratio.
ω	4	Non-feasible solution penalty constant.
γ	1	Pheromone change constant.
Number of Ants	$\lvert DQ\rvert + \lvert S\rvert$	The number of ants.
τ_0	0.1	The amount of pheromone initially on each edge in the graph.
m	$\lfloor$# Ants $\cdot\ 0.2\rfloor$	The top m ants are allowed to deposit pheromone.
p	$\lfloor$# Ants $\cdot\ 0.1\rfloor$	The top p ants are allowed to update policy.
δ_l	0.03	Losing step size.
δ_w	0.005	Winning step size.

Table 7: Parameter and condition values for the WoLFAntDA and PD-WoLFAntDA experiments.

were determined in the same manner as AntDA with the exception that there are a few new variables: δ_l, δ_w, and p. The main focus was on tuning these new parameters, but tests were also run for α, β, and m to determine their best values for WoLFAntDA and PD-WoLFAntDA. Through this process, unless otherwise stated, these two algorithms are run with the parameter values and conditions shown in Table 5.

WoLFAntDA and PD-WoLFAntDA are also fairly insensitive to a change in the values shown in Table 5. However, just as in AntDA, the parameter that seemed to have the biggest impact was the number of ants allowed to change edge pheromone values. The impacts of this parameter, m, are presented in section 4.6.

4.4 *Comparison of AntDA To Other Optimization Algorithms*

Tables 8 and 9 show the performance of AntDA and the other solution methods for forty-five test cases. Each row of the tables represents one test case while columns group each solution method. The Random column shows the lowest-cost solution produced over 1000 executions of the Random algorithm. For the ant-based results, the minimum, maximum, average, and standard deviation of the fifty solutions for each test case are shown. Recall that AntDA is run 50 times for each test case and that each running is for 400 time steps. The lowest-cost solutions for each test case are shown in **bold** typeface.

AntDA found the solution with the lowest update burden in all but three test cases. Also, in all but two cases, the solution with the maximum update burden found by AntDA is better than the minimum update burden found by the Random and Greedy solution methods.

Clearly, AntDA produces better solutions than the three other methods. However, AntDA has higher solution times than the other methods. For example, in the 5 DA, 3 quality, increasing RL pattern, high UL pattern experiments, the Greedy algorithm can produce a solution in milliseconds, the Random algorithm needed about 1.5 minutes, and LINGO was cut off after two weeks. Yet, on the same hardware, AntDA requires an average of 7.2 minutes to complete the 400 time steps and produce a single solution. Since AntDA was run 50 times, its run-time was close to 6 hours. For more complex problems, AntDA took as long as a week to run through 50 times. However, 400 time steps and 50 runnings is being overly thorough. Reducing the number of time steps would allow for much faster results. The number of time steps (or convergence rate) necessary to AntDA find the best solution is presented in section 4.7.

4.5 *Effect of the Server Filling Algorithm*

This section highlights the impact of the Server-Filling (SF) optimization heuristic. Figure 10 shows the minimum update burdens produced by AntDA, WoLFAntDA, and PD-WoLFAntDA with and without the Server-Filling heuristic.

# DA	Quals Per DA	RL Patt	UL Patt	#	Solution Cost (Update Burden)							# Ants
								AntDA				
					Random	Greedy	LINGO	min	max	avg	stdev	
				1	305	258	246^{+}	**239**	249	245.68	1.67	45
				2	254	225	**203**$^{+}$	**203**	207	203.72	0.97	45
5	1	n/a	low	3	298	243	**231**$^{+}$	**231**	231	231	0.0	50
				4	379	323	313^{+}	**305**	314	308.22	2.40	50
				5	351	307	292^{+}	**284**	292	287.06	2.27	45
				1	930	860	821^{+}	**796**	821	799.16	4.70	60
				2	698	659	656^{+}	**645**	647	645.24	0.66	55
5	1	n/a	high	3	895	894	**856**$^{+}$	**856**	945	916	24.16	55
				4	998	983	964^{+}	**953**	1009	976.1	19.48	55
				5	810	761	710^{+}	**708**	727	713.06	4.25	55
				1	259	211	206*	**196**	201	197	1.81	50
				2	188	164	166*	**160**	160	160	0.0	50
5	2	decr	low	3	271	226	230	**217** *	220	217.78	1.25	55
				4	169	157	156*	**155**	155	155	0.0	50
				5	230	194	193*	**187**	188	187.02	0.14	50
				1	1070	890	**829***	838	850	849.12	2.39	65
				2	990	909	**831***	838	850	844.8	3.11	60
5	2	incr	high	3	999	819	**781***	786	813	796.74	9.17	65
				4	1167	957	1002*	**858**	860	858.08	0.40	65
				5	974	809	832*	**720**	728	722.58	2.89	65
				1	242	206	237^{++}	**178**	179	178.02	0.14	55
				2	220	186	215^{++}	**158**	159	158.04	0.20	55
5	3	decr	low	3	186	155	166^{++}	**154**	155	154.07	0.27	55
				4	177	151	158^{++}	**142**	142	142.00	0.00	55
				5	196	171	176^{++}	**145**	147	146.40	0.57	55
				1	1057	842	961**	**784**	800	793.86	5.77	70
				2	1135	884	940**	**811**	824	817.94	2.94	70
5	3	incr	high	3	1048	788	907**	**764**	771	766.06	2.78	70
				4	1099	849	885**	**813**	822	818.98	2.02	75
				5	1137	867	913**	**811**	823	814.64	2.60	70

Table 8: Comparison of AntDA to other search algorithms for 5 DA problems. Depending on the Problem, LINGO was let run for differing amounts of time: (+) = 1 hour, (*) = 2 hours, (++) = 4 hours, and (**) = 300 hours. The lowest-cost solutions for each test case are shown in **bold** typeface.

Server-Filling experimental results are shown for just five test cases (hypothetical DA*rep* instances) involving five DAs of three quality levels each with an increasing request load pattern and a high update load pattern. These test cases were chosen because they proved to be the most difficult to solve in a reasonable amount of time.

Using SF reduced the update burden by an average of 39.0 points for all three algorithms which translates to a 4.64% reduction on average for these 5 test cases. In all

# DA	Quals Per DA	RL Patt	UL Patt	#	Random	Greedy	AntDA min	AntDA max	AntDA avg	AntDA stdev	# Ants
10	3	decr	low	1	478	336	**328**	335	330.5	1.52	105
				2	532	360	**340**	344	342.38	1.28	110
				3	489	332	**325**	326	325.24	0.43	110
				4	492	324	**318**	321	319.58	0.76	110
				5	476	320	**309**	311	310.9	0.36	110
10	3	incr	high	1	2401	1703	**1635**	1644	1639.71	2.90	135
				2	2547	1817	**1719**	1749	1734.93	7.87	140
				3	2335	1676	**1565**	1600	1584.73	10.30	140
				4	2669	1899	**1796**	1826	1812.90	6.70	140
				5	2531	1791	**1711**	1734	1725.31	5.41	135
20	3	decr	low	1	1129	717	**690**	695	692.76	1.19	215
				2	1074	705	**677**	687	681.96	2.16	210
				3	1081	695	**676**	685	681.66	2.03	210
				4	1069	699	**667**	676	670.82	2.98	215
				5	1137	756	**725**	734	729.88	2.24	215

Table 9: Comparison of AntDA to other search algorithms for 10 and 20 DA problems. The lowest-cost solutions for each test case are shown in **bold** typeface.

#	AntDA Server Filling Off	AntDA Server Filling On	AntDA % Diff	WoLFAntDA Server Filling Off	WoLFAntDA Server Filling On	WoLFAntDA % Diff	PD-WoLFAntDA Server Filling Off	PD-WoLFAntDA Server Filling On	PD-WoLFAntDA % Diff
1	823	784	4.74	832	797	4.21	827	793	4.11
2	841	811	3.57	862	819	4.99	854	814	4.68
3	793	764	3.66	806	764	5.21	793	764	3.66
4	852	813	4.58	851	818	3.88	864	816	5.56
5	852	811	4.81	870	814	6.44	861	814	5.46
	Average		4.272	Average		4.946	Average		4.694

Table 10: The Effect of the server filling algorithm on AntDA and WoLFAntDA.

cases, SF improves solution values. This demonstrates the importance of local heuristics (non-ACO) for improving performance of ACO algorithms (as was also seen in other ACO work [34, 38]).

4.6 Effect of Only Allowing the Top m of Ant Solutions to Deposit Pheromone

This section discusses the impact of only allowing the ants with the top m solutions to deposit pheromone after each iteration. Figures 7 and 8 show the average update burden

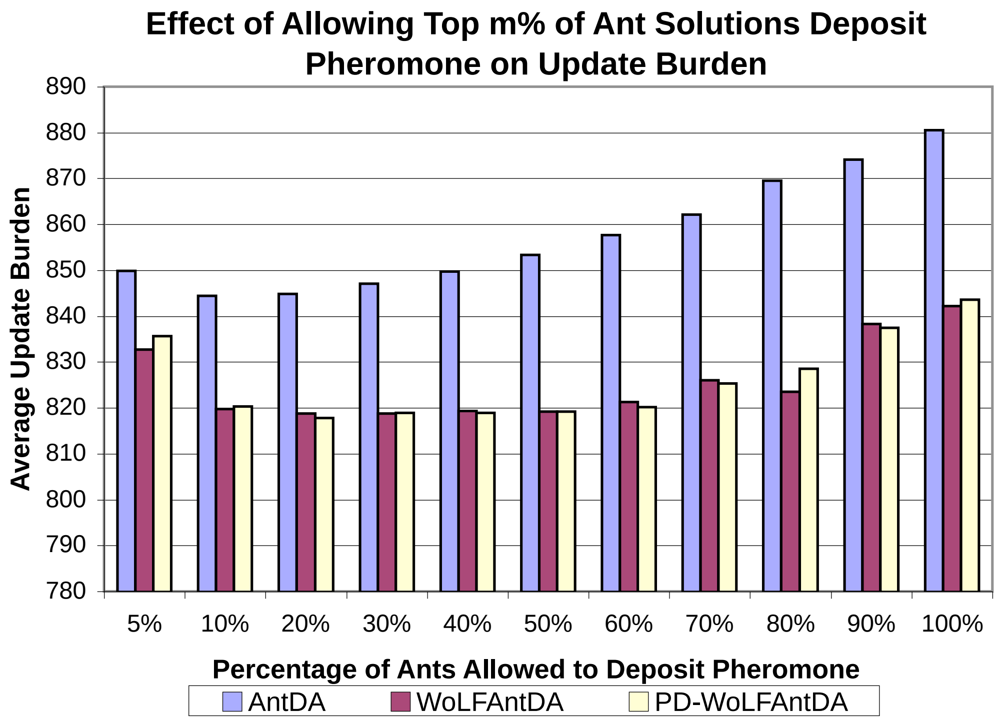

Figure 7: Effect of limiting the number of ants allowed to deposit pheromone on update burden. Smaller update burdens are better.

and convergence rates produced by AntDA and both versions of WoLFAntDA for many different percentages of ants allowed to deposit pheromone.

Experimental results are shown for just one test case (Problem #5 of the test cases with five DAs of three quality levels each with an increasing request load pattern and a high update load pattern). This test case is chosen because it proved to be the most difficult to solve in a reasonable amount of time, however results are representative of all test cases.

For AntDA, only allowing the ants with the top 10% of solutions to deposit pheromone after each iteration versus allowing all ants to deposit pheromone decreased the average update burden experienced by 36.1 points while decreasing the convergence rate by 141.16 time steps. This translates to a decrease in average update burden of 4.1% while cutting the convergence rate by 4.6 times.

For WoLFAntDA and PD-WoLFAntDA, allowing only ants with the top 20% of solutions to deposit pheromone after each iteration versus allowing all ants to deposit

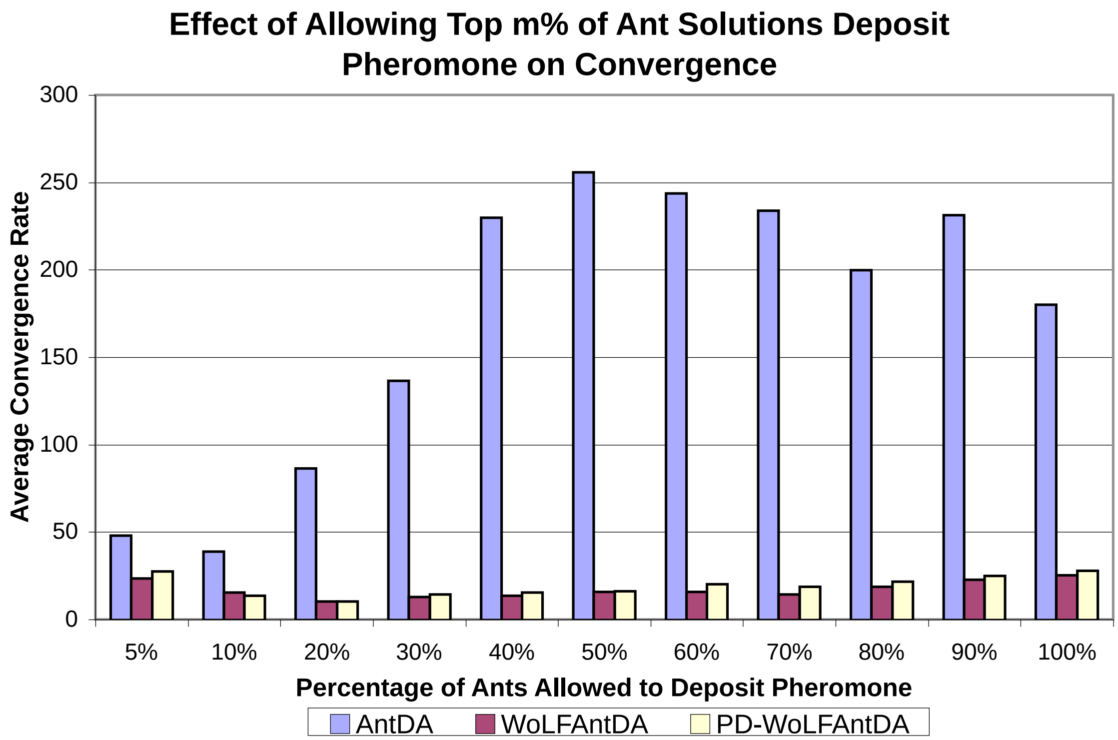

Figure 8: Effect of limiting the number of ants allowed to deposit pheromone on convergence. Smaller convergence rates are better.

pheromone didn't have as big of an impact (probably due to the addition of edge policy values). However, it led to a decrease of over 24 points in average update burden experienced and still cut convergence by over 2.5 times.

4.7 Comparison of WoLFAntDA and PD-WoLFAntDA To AntDA

Tables 11 and 12 show the performance of AntDA, WoLFAntDA, and PD-WoLFAntDA for forty-five test cases. Each row of the tables represents one test case. For each test case, the minimum, average, and standard deviation of the fifty solutions are shown. The lowest-cost solutions for each test case are shown in **bold** typeface.

As seen in the table, WoLFAntDA and PD-WoLFAntDA find solutions comparable to AntDA for all of the five DA test cases. However, with the rise of complexity in the ten and twenty DA test cases (Table 12), WoLFAntDA and PD-WoLFAntDA begin to lose ground on AntDA, this could be due to the fact that parameter selection was performed

# DA	Quals Per DA	RL Patt	UL Patt	#	Solution Cost (Update Burden) AntDA min			WoLFAntDA			PD-WoLFAntDA		
					min	avg	stdev	min	avg	stdev	min	avg	stdev
5	1	n/a	low	1	**239**	245.68	1.67	241	250	3.06	241	245.68	2.26
				2	**203**	203.72	0.97	**203**	206.26	1.74	214	206.14	1.64
				3	**231**	231	0.0	**231**	233.02	1.04	**231**	232.9	2.09
				4	305	308.22	2.40	**304**	312.12	3.13	306	312.48	21.3
				5	**284**	287.06	2.27	**284**	290.78	3.45	**284**	288.26	2.78
5	1	n/a	high	1	**796**	799.16	4.70	800	824.92	10.88	**796**	814.20	13.83
				2	**645**	645.24	0.66	**645**	649.06	6.61	**645**	647.50	5.55
				3	**856**	916.00	24.16	874	936.66	37.18	**856**	920.62	29.25
				4	953	976.1	19.48	969	1010.58	13.43	**951**	992.56	19.71
				5	**708**	713.06	4.25	717	733.36	7.39	**708**	727.32	9.68
5	2	decr	low	1	**196**	197	1.81	200	202.26	2.29	**196**	202.76	2.47
				2	**160**	160	0.0	**160**	160	0.0	**160**	160.04	0.28
				3	**217**	217.78	1.25	219	220.72	0.95	**217**	220.34	1.12
				4	**155**	155	0.0	**155**	155.58	0.50	**155**	155.4	0.49
				5	**187**	187.02	0.14	**187**	190.06	1.19	**187**	189.96	1.34
5	2	incr	high	1	**838**	849.12	2.39	**838**	855.4	7.94	**838**	852.82	7.31
				2	**838**	844.80	3.11	855	865.58	6.70	850	862.62	6.32
				3	**786**	796.74	9.17	**786**	811.48	8.06	**786**	805.30	8.90
				4	**858**	858.08	0.40	**858**	867.00	8.06	**858**	861.58	5.16
				5	**720**	722.58	2.89	721	728.64	4.42	**720**	724.68	3.69
5	3	decr	low	1	**178**	178.02	0.14	**178**	179.16	1.22	**178**	178.90	1.09
				2	**158**	158.04	0.20	**158**	160.88	2.02	**158**	160.32	1.99
				3	**154**	154.07	0.27	155	157.11	1.64	155	157.69	1.58
				4	**142**	142.00	0.00	**142**	144.34	1.44	**142**	144.86	1.75
				5	**145**	146.40	0.57	147	148.32	1.41	146	147.84	1.28
5	3	incr	high	1	**784**	793.86	5.77	797	809.06	5.99	793	806.30	5.99
				2	**811**	817.94	2.94	819	838.04	8.99	814	832.54	8.79
				3	**764**	766.06	2.78	**764**	776.58	6.44	**764**	776.54	6.04
				4	**813**	818.98	2.02	818	825.32	2.97	816	824.14	3.05
				5	**811**	814.64	2.60	814	828.04	7.50	814	823.38	8.23

Table 11: Comparison of update burden for WoLFAntDA and PD-WoLFAntDA to AntDA for 5 DA problems. The lowest-cost solutions for each test case are shown in **bold** typeface.

for the five DA problems and not re-calibrated for larger problems. The policy values could be increasing too rapidly on some edges and slightly tweaking the δ_l and δ_w values could cause the ants to explore more. However, in the problem with the largest difference between AntDA and the other two algorithms (the 20 DA test cases), AntDA's minimum update burden found was less than 3% lower than the minimum update burden found by both WoLFAntDA and PD-WoLFAntDA. In other words, the difference is small. In five of the trials, AntDA or WoLFAntDA found their best solutions with a standard deviation of zero. This improbable behavior occurs when the algorithm finds the same best answer after 400 iterations every time.

A two-tailed t-test was conducted to ensure that WoLFAntDA and PD-WoLFAntDA's solution values were statistically different from AntDA's solutions. Using an alpha level of

# DA	Quals Per DA	RL Patt	UL Patt	#	Solution Cost (Update Burden) AntDA min	avg	stdev	WoLFAntDA min	avg	stdev	PD-WoLFAntDA min	avg	stdev
10	3	decr	low	1	**328**	330.50	1.52	332	336.71	1.66	333	336.32	1.62
				2	**340**	342.38	1.28	349	354.74	1.82	350	354.59	2.16
				3	**325**	325.24	0.43	329	330.10	0.67	328	329.93	0.83
				4	**318**	319.58	0.76	321	324.03	1.51	321	323.89	1.74
				5	**309**	310.90	0.36	312	316.89	1.99	313	316.69	1.62
10	3	incr	high	1	**1635**	1639.71	2.90	1659	1677.03	8.61	1656	1674.86	7.58
				2	**1719**	1734.93	7.87	1758	1784.56	13.18	1735	1779.79	16.85
				3	**1564**	1584.73	10.30	1604	1635.29	11.64	1612	1629.56	10.92
				4	**1796**	1812.90	6.70	1832	1858.06	9.11	1828	1857.05	9.77
				5	**1711**	1725.31	5.41	1747	1772.56	10.29	1725	1768.41	10.76
20	3	decr	low	1	**690**	692.76	1.19	705	711.62	3.43	706	710.42	2.99
				2	**677**	681.96	2.16	690	700.00	4.37	697	700.31	2.21
				3	**676**	681.66	2.03	687	693.07	2.62	690	693.75	2.38
				4	**667**	670.82	2.98	677	684.10	3.63	679	684.60	2.99
				5	**725**	729.88	2.24	745	747.50	1.96	744	746.60	1.43

Table 12: Comparison of update burden for WoLFAntDA and PD-WoLFAntDA to AntDA for 10 and 20 DA problems. The lowest-cost solutions for each test case are shown in **bold** typeface.

0.05 and a degree of freedom of 98 ((50 trials for WoLFAntDA or PD-WoLFAntDA -1)(50 trials for AntDA -1)), it was found that in all cases the probability that there is no difference between the means is less than 0.05. Therefore, making the results for update burden and convergence rates of WoLFAntDA and PD-WoLFAntDA statistically significant compared to the results of AntDA.

Tables 13 and 14 display the convergence rates for the three algorithms for all 45 test cases. In the 1 quality test cases, WoLFAntDA and PD-WoLFAntDA converged at approximately the same rate as AntDA. However, as the problem complexity rose, WoLFAntDA and PD-WoLFAntDA converged at a much faster rate, leading to a decrease in the average convergence rate of 99.13% in the 20 DA test cases.

Although WoLFAntDA and PD-WoLFAntDA produce solutions that are worse than AntDA, the solutions are still better than the solutions found by the other solution methods, as seen in tables 15 and 16. A speedup in convergence of over 99% for less than a 3% decline in solution cost makes WoLFAntDA and PD-WoLFAntDA much more competitive with other solution methods. Using the rule-of-thumb that 95% of values fall within two standard deviations of their mean, this means that for the 20 DA test cases, AntDA would have to be run for 385 time steps ($149 + 2 \cdot 117.82$) to have a 95% confidence that it is

# DA	Quals Per DA	RL Patt	UL Patt	#	Convergence Rate (Iterations) AntDA min	avg	stdev	WoLFAntDA min	avg	stdev	PD-WoLFAntDA min	avg	stdev
5	1	n/a	low	1	1	3.40	2.02	1	3.80	3.04	1	**3.32**	2.80
				2	1	8.06	3.74	1	**6.72**	10.40	1	9.8	14.21
				3	2	4.7	1.18	1	3.9	2.31	1	**3.3**	2.71
				4	1	14.02	5.16	1	5.3	7.14	1	**3.92**	5.65
				5	1	7.96	3.48	1	**3.54**	2.05	1	5.72	3.55
5	1	n/a	high	1	10	15.40	5.43	2	**6.86**	4.57	4	9.26	4.78
				2	2	7.02	2.43	1	**5.14**	2.81	2	5.78	3.97
				3	1	**13.46**	32.43	1	37.38	62.46	1	37.46	70.14
				4	1	29.92	64.25	1	**17.34**	46.52	1	41.36	60.70
				5	10	17.24	5.92	2	**10.46**	7.70	2	14.44	12.91
5	2	decr	low	1	1	8.9	22.34	1	**1.06**	0.24	1	1.36	1.06
				2	1	2.44	0.78	1	**1.70**	0.46	1	1.84	0.71
				3	1	11.40	8.49	1	**1.28**	0.57	1	1.60	1.05
				4	2	3.82	1.93	1	2.80	8.60	1	**1.80**	1.05
				5	3	10.12	4.21	1	2.12	2.25	1	**1.66**	1.59
5	2	incr	high	1	2	9.66	10.74	1	**5.26**	2.63	2	6.00	2.04
				2	7	110.56	126.70	1	**2.1**	2.07	1	2.52	2.33
				3	1	19.56	47.66	1	**4.16**	2.54	1	7.08	4.62
				4	2	6.42	4.83	1	**3.88**	2.37	1	4.18	2.32
				5	2	16.34	29.69	1	**5.34**	3.27	2	7.16	2.50
5	3	decr	low	1	2	25.3	72.84	1	**1.68**	0.89	1	2.04	1.12
				2	2	6.62	3.23	1	**1.98**	1.15	1	2.96	3.73
				3	5	10.98	10.63	1	**2.24**	1.86	1	2.38	1.19
				4	2	5.32	4.24	1	2.04	4.95	1	**1.4**	1.01
				5	1	18.14	56.23	1	2.80	9.29	1	**2.56**	6.73
5	3	incr	high	1	3	13.38	11.52	1	**3.40**	3.12	1	5.44	4.33
				2	4	20.82	11.21	1	**5.00**	2.99	1	7.66	3.93
				3	1	5.36	3.50	1	3.14	2.46	1	**2.64**	2.36
				4	4	34.74	63.95	1	**3.40**	2.33	1	6.22	10.08
				5	7	16.58	27.56	1	**5.96**	2.86	2	8.70	3.66

Table 13: Comparison of convergence rates for WoLFAntDA and PD-WoLFAntDA to AntDA for 5 DA problems. The lowest average convergence rate for each test case are shown in **bold** typeface.

finding the best solutions possible. With an average of a minute per time step for 20 DA problems, it would take AntDA almost 6.5 hours to find its best solution. Using either WoLFAntDA or PD-WoLFAntDA, for the same test cases, it requires runs of a mere 2 time steps ($1.22 + 2 \cdot 0.383$). It would therefore only take WoLFAntDA or PD-WoLFAntDA two minutes to find its best solution.

It is interesting to note that test cases with a *high* update load have higher standard deviations for cost (Tables 11 and 12) and convergence (Tables 13 and 14). This is due to

# DA	Quals Per DA	RL Patt	UL Patt	#	Convergence Rate (Iterations)								
					AntDA			WoLFAntDA			PD-WoLFAntDA		
					min	avg	stdev	min	avg	stdev	min	avg	stdev
10	3	decr	low	1	4	210.88	110.76	1	1.22	1.56	1	**1.05**	0.21
				2	6	172.12	110.24	1	**1.13**	0.34	1	1.20	0.46
				3	20	130.58	91.59	1	**1.45**	0.50	1	1.61	0.58
				4	3	16.64	23.82	1	**1.03**	0.16	1	1.26	0.44
				5	3	52.08	68.75	1	**1.07**	0.25	1	1.33	0.61
10	3	incr	high	1	61	137.39	49.81	1	**1.83**	0.86	1	2.86	2.66
				2	19	221.00	86.70	1	**6.92**	4.71	1	8.54	6.09
				3	21	203.65	102.68	1	**4.32**	3.45	2	6.85	5.11
				4	11	159.48	68.74	1	**1.83**	1.56	1	2.57	2.35
				5	22	217.37	74.73	1	**2.54**	1.89	1	2.61	3.33
20	3	decr	low	1	10	195.84	112.06	1	1.46	0.52	1	**1.25**	0.45
				2	9	186.38	114.72	1	**1.08**	0.28	1	1.31	0.48
				3	5	126.46	120.26	1	1.14	0.36	1	**1.00**	0.00
				4	6	98.29	123.40	1	1.20	0.42	1	**1.10**	0.32
				5	5	137.70	118.67	1	1.40	0.52	1	**1.30**	0.48

Table 14: Comparison of convergence rates for WoLFAntDA and PD-WoLFAntDA to AntDA for 10 and 20 DA problems. The lowest average convergence rate for each test case are shown in **bold** typeface.

the fact that in the high UL pattern, update loads are boosted so that some servers are not able to host high-quality replicas of some DAs. This causes more difficulty in solving the DA*rep* problem therefore causing a bigger disparity in solutions and convergence rates.

There were no big differences in the performances of WoLFAntDA and PD-WoLFAntDA. Although WoLFAntDA had the lowest average in convergence rate more often, PD-WoLFAntDA was never far off. Additionally, each found approximately the same update burden for each test case. This shows that both rules for determining whether an agent is *winning* or *losing*, once tuned correctly, can be used to effectively solve the Quality-Sensitive DA Replication Problem.

# DA	Quals Per DA	RL Patt	UL Patt	#	Solution Cost (Update Burden) Random	Greedy	LINGO	WoLF-AntDA	PD-WoLF-AntDA	AntDA
5	1	n/a	low	1	305	258	246^{+}	**239**	241	241
				2	254	225	**203**$^{+}$	**203**	**203**	214
				3	298	243	**231**$^{+}$	**231**	**231**	**231**
				4	379	323	313^{+}	305	**304**	306
				5	351	307	292^{+}	**284**	**284**	**284**
5	1	n/a	high	1	930	860	821^{+}	**796**	800	**796**
				2	698	659	656^{+}	**645**	**645**	**645**
				3	895	894	**856**$^{+}$	**856**	874	**856**
				4	998	983	964^{+}	953	969	**951**
				5	810	761	710^{+}	**708**	717	**708**
5	2	decr	low	1	259	211	206*	**196**	200	**196**
				2	188	164	166*	**160**	**160**	**160**
				3	271*	226	230	**217**	219	**217**
				4	169	157	156*	**155**	**155**	**155**
				5	230	194	193*	**187**	**187**	**187**
5	2	incr	high	1	1070	890	**829***	838	838	838
				2	990	809	**831***	838	855	850
				3	999	819	**781***	786	786	786
				4	1167	957	1002*	**858**	**858**	**858**
				5	974	809	832*	**720**	721	**720**
5	3	decr	low	1	242	206	237^{++}	**178**	**178**	**178**
				2	220	186	215^{++}	**158**	**158**	**158**
				3	186	155	166^{++}	**154**	155	155
				4	177	151	158^{++}	**142**	**142**	**142**
				5	196	171	176^{++}	**145**	147	146
5	3	incr	high	1	1057	842	961**	**784**	797	793
				2	1135	884	940**	**811**	819	814
				3	1048	788	907**	**764**	**764**	**764**
				4	1099	849	885**	**813**	818	816
				5	1137	867	913**	**811**	814	814

Table 15: Comparison of AntDA, WoLFAntDA, and PD-WoLFAntDA to other search algorithms for 5 DA problems. Depending on the Problem, LINGO was let run for differing amounts of time: (+) = 1 hour, (*) = 2 hours, (++) = 4 hours, and (**) = 300 hours. Only the best solution found for each method are shown. The best solutions for each problem instance are shown in **bold** typeface.

# DA	Quals Per DA	RL Patt	UL Patt	#	Solution Cost (Update Burden) Random	Greedy	WoLF-AntDA	PD-WoLF-AntDA	AntDA
10	3	decr	low	1	478	336	**328**	332	333
				2	532	360	**340**	349	350
				3	489	332	**325**	329	328
				4	492	324	**318**	321	321
				5	476	320	**309**	312	313
10	3	incr	high	1	2401	1703	**1635**	1659	1656
				2	2547	1817	**1719**	1758	1735
				3	2335	1676	**1565**	1604	1612
				4	2669	1899	**1796**	1832	1828
				5	2531	1791	**1711**	1747	1725
20	3	decr	low	1	1129	717	**690**	705	706
				2	1074	705	**677**	690	697
				3	1081	695	**676**	687	690
				4	1069	699	**667**	677	679
				5	1137	756	**725**	745	744

Table 16: Comparison of AntDA WoLFAntDA and PD-WoLFAntDA to other search algorithms for 10 and 20 DA problems. Only the best solution found for each method are shown. The best solutions for each problem instance are shown in **bold** typeface.

V. Conclusion

This thesis effort examined several aspects of the Quality-Sensitive DA Replication Problem (DA*rep*). In this problem, the ASP must assign DA replicas to its network of heterogeneous servers so that user demand is satisfied at the desired quality level and replica update loads are minimized. It then proposed three simple algorithms for solving it, and validated and analyzed the performance of the proposed algorithms compared to other search algorithms.

5.1 Contributions and Achievements

Major accomplishments and achievements of this thesis investigation include the following.

1. DA*rep* is thoroughly discussed and the practical impediments to its solution were identified (Chapter II).
2. Problems similar to DA*rep* were reviewed and reasons why solutions to those problems are ill-suited for DA*rep* are discussed (Chapter II).
3. The ant colony optimization (ACO) meta-heuristic is discussed and has been shown to be successful in solving difficult discrete optimization problems (Chapter II). An ant colony algorithm, AntDA, originally proposed in [53], is further investigated and results on its performance reported. Highlights in the AntDA investigation include the following (Chapter IV).
   (a) Better values for the tunable parameters were determined and were shown to lead AntDA to better solutions and faster convergence.
   (b) Limiting the number of ants depositing pheromone at the end of a time step found that only allowing the ants with the top 10% of solutions to deposit pheromone led to a convergence rate of over 4.5 times faster than allowing all ants to deposit pheromone while reducing update burden by 4%. This occurs due to the reinforcement of the edges which are used in the best solutions which allowed the ants to search in the area of good solutions.

(c) Ants are allowed to invoke a Server Filling replica creation policy when creating a replica on a server. This led to a reduction in update burden by more than 4.5% on average. It does this by assigning additional qualities of a $\langle d, q \rangle$ pair to a server with remaining capacity which keeps system-wide update burden low.

4. In order to help AntDA converge quicker and find better solutions to more complex problems, it was combined with the variable-step policy hill-climbing algorithm called Win or Learn Fast (WoLF) to create two algorithms, WoLFAntDA and PD-WoLFAntDA. Both algorithms are discussed (Chapter III) and results on their performance reported (Chapter IV). Highlights in the WoLFAntDA and PD-WoLFAntDA investigation include the following.

   (a) Better values for the tunable parameters are determined for the DA*rep* problem and are shown to lead WoLFAntDA and PD-WoLFAntDA to converge very rapidly with better solutions.

   (b) Limiting the number of ants depositing pheromone at the end of a time step was also experimented with for WoLFAntDA and PD-WoLFAntDA. This time, it was found that only allowing the ants with the top 20% of solutions to deposit pheromone led to a decrease in convergence rate of over 2.5 times compared to allowing all ants to deposit pheromone.

   (c) The number of ants allowed to update an edge's policy values was also experimented with. Results show that only allowing the ants with the top 10% of solutions to update policy values led to a decrease in convergence rate while keeping update burden below other solution methods.

   (d) WoLFAntDA and PD-WoLFAntDA allowed the convergence rates to solve the most complex problem to be decreased by over 99% compared to AntDA while only finding solution values of less than 3% higher on average.

   (e) The addition of the learning algorithm into AntDA, allowed the ACO heuristic to be applied to more complex problems while still being able to be solved in a reasonable amount of time. With WoLFAntDA or PD-WoLFAntDA, a 20 DA,

3 quality problem (the hardest tested) could find a better solution than the other search algorithms tested by the second iteration on average. Therefore, bringing its run time down to just two minutes, instead of hours with AntDA.

5.2 Future AntDA Work

Incorporating the learning algorithm into AntDA allowed WoLFAntDA to address the issue of scalability, which is one of the most significant drawbacks of ant-based algorithms. It allowed AntDA to be useful for realistically-sized problems (20 DAs) by improving convergence rates but failed to converge to the best answers found by AntDA. In this regard, further research and testing needs to be done. The main goal of any further work should be to get the AntDA algorithm to keep the convergence rate of WoLFAntDA and PD-WoLFAntDA, but to improve solution values. The following list describes some ideas for future work.

1. Starting ants at an artificial start node (ants learn which $\langle d, q \rangle$ pair to assign first), instead of positioning them at a randomly chosen server vertices, may have an impact. However, since the halting criteria is examined when moving from S to DQ vertices, starting ants at a server vertex is probably wise.
2. Implement Equation 19 (Chapter III) so that the top-scoring solutions deposit proportionally more pheromone on edges than low-scoring solutions.
3. Implement Equation 28 (Chapter III) so that the top-scoring solutions can add or subtract proportionally more policy on edges than low-scoring solutions.
4. Develop a visualization tool so that the graph state and algorithm behavior can be monitored. Such a tool would allow the researcher to better examine the impact of parameter values and algorithmic problem areas that could be further addressed.
5. Adapt AntDA, WoLFAntDA, and PD-WoLFAntDA for use in dynamic environments.
6. Allow AntDA, WoLFAntDA, and PD-WoLFAntDA to run on better hardware and be applied to more complex problems.

7. Compare AntDA, WoLFAntDA, and PD-WoLFAntDA with other stochastic techniques such as breadth-first search, depth-first search in order to ensure the worth of these three algorithms in solving the Quality-Sensitive DA Replication Problem.

8. Adapt AntDA, WoLFAntDA, and PD-WoLFAntDA to be run in parallel to allow for faster results. These algorithms are essentially in this form already (since each ant solves on its own for each time step on a map that is only updated between time steps and redistributed).

9. Finally, future work should include the application of ACO and WoLF in combination to other algorithms such as Quadratic Assignment Problem (QAP), the Vehicle Routing Problem, and many other problems which ACO has already been applied.

Bibliography

1. Aerts, Joep, Jan Korst, and Frits C. R. Spieksma. "An Approximation Algorithm for a Generalized Assignment Problem with Small Resource Constraints". 2003.
2. Akamai Technologies. Http://www.akamai.com.
3. Alaya, Inès, Christine Solnon, and Khaled Ghédira. "Ant Algorithm for the Multidimensional Knapsack Problem". *International Conference on Bioinspired Optimization Methods and their Applications*, 63–72. 2004. URL `www710.univ-lyon1.fr/ csolnon/publications/bioma04.pdf`.
4. Amar, L., A. Barak, and A. Shiloh. "The MOSIX Direct File System Access Method for Supporting Scalable Cluster File Systems". *Cluster Computing*, 7(2):141–150, 2004.
5. Amir, Yair, Baruch Awerbuch, Amnon Barak, R. Sean Borgstrom, and Arie Keren. "An Opportunity Cost Approach for Job Assignment in a Scalable Computing Cluster". *IEEE Transactions on Parallel and Distributed Systems*, 11(7):760–768, 2000.
6. ASP-One. Http://www.asp-one.com.
7. Balachandran, K. R. "Purchasing Priorities in Queues". *Management Science*, 18:319–326, 1972.
8. Banerjee, B. and J. Peng. "Adaptive policy gradient in multiagent learning", 2003. URL `citeseer.ist.psu.edu/banerjee03adaptive.html`.
9. Bonabeau, E., M. Dorigo, and T. Théraulaz. *From Natural to Artificial Swarm Intelligence*. Oxford University Press, New York, 1999.
10. Bonabeau, E., F. Henaux, S. Guérin, D. Snyers, P. Kuntz, and G. Théraulaz. "Routing in telecommunication networks with "Smart" ant-like agents telecommunication applications". *Lectures Notes in AI*, 1437:60–71, 1998.
11. Bowling, Michael. "Multiagent Learning in the Presence of Limited Agents". 2005.
12. Bowling, Michael and Manuela Veloso. "Rational and Convergence Learning in Stochastic Games".
13. Bright, L. and L. Raschid. "Using Latency-Recency Profiles for Data Delivery on the Web". *VLDB*, 550–561. 2002.
14. Bullnheimer, B., R. F. Hartl, and C. Strauss. "A New Rank Based Version of the Ant System: A Computational Study". *Central European Journal for Operations Research and Economics*, 7(1):25–38, 1999.
15. Bullnheimer, Bernd, Richard Hartl, and Christine Strauss. "An improved Ant System algorithm for the Vehicle Routing Problem". *Annals of Operations Research*, 89:319–328, 1999. Available from Kluwer Online via MQ library.

16. Burkard, R. E. “Locations with Spacial Interactions: The Quadratic Assignment Problem”. *Discrete Location Theory*. 1990.

17. Candan, K. S., D. Agrawal, W. S. Li, O. Po, and W. Hsiung. “View Invalidation for Dynamic Content Caching in Multitiered Architectures”. *VLDB*, 562–573. 2002.

18. Candan, K. S. and W. Li. *Architectural Issues of Web-Enabled Electronic Business*, chapter Integration of Database and Internet Technologies for Scalable End-to-end E-commerce Systems, 84–112. Idea Group Publishing, 2002.

19. Candan, K. Selçuk, W. S. Li, Qiong Luo, W. P. Hsiung, and Divyakant Agrawal. “Enabling Dynamic Content Caching for Database-Driven Web Sites”. *SIGMOD*, 532–543. ACM, 2001.

20. Carney, Donald, Sangdon Lee, and Stanley B. Zdonik. “Scalable Application-Aware Data Freshening”. *ICDE*, 481–492. 2003.

21. Challenger, Jim, Paul Dantzig, and Arun Iyengar. “A Scalable System for Consistently Caching Dynamic Web Data”. *INFOCOM*, 294–303. 1999. URL `citeseer.nj.nec.com/article/challenger99scalable.html`.

22. Cherniack, M., E. F. Galvez, M. J. Franklin, and S. Zdonik. “Profile-Driven Cache Management”. *ICDE*, 645–656. 2003.

23. Colorni, A., M. Dorigo, V. Maniezzo, and M. Trubian. “Ant system for Job-shop Scheduling”. *Belgian Journal of Operations Research, Statistics and Computer Science*, 34(1):39–53, 1994.

24. Comellas, Francesc and Javier Ozón. “An Ant Algorithm for the Graph Colouring Problem”. *First International Workshop on Ant Colony Optimization (ANTS ’98)*. 1998.

25. Costa, D. and A. Hertz. “Ants Can Colour Graphs”. *Journal of the Operational Research Society*, 48:295–305, 1997.

26. Cousin, K, Gilbert L. Peterson, Gary B. Lamont, and Christopher B. Mayer. “WoLF Ants”. 2006.

27. Cousin, Kevin and Gilbert L. Peterson. “Cooperative Reinforcement Learning Using an Expert Measuring Weighted Strategy With WoLF”.

28. Datta, A., K. Dutta, H. Thomas, D. VanderMeer, Suresha, and K. Ramamritham. “Proxy-Based Acceleration of Dynamically Generated Content on the World Wide Web: An Approach and Implementation”. *Transactions on Database Systems*, 29(2):403–443, 2004.

29. Di Caro, G. and M. Dorigo. “AntNet: Distributed stigmergetic control for communications networks”. *Journal of Artificial Intelligence Research*, 9:317–365, December 1998.

30. Di Caro, G. and M. Dorigo. "Two ant colony algorithms for best-effort routing in datagram networks". *IASTED International Conference on Parallel and Distributed Computing and Systems*, 541–546. 1998.

31. Diaz, J. A. *Algorithmic Approaches for the Single Source Capacitated Plant Location Problem*. Ph.D. thesis, Universitat Politecnica de Catalunya, Barcelona, 2001.

32. Dorigo, M. and L. M. Gambardella. "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem". *Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

33. Dorigo, M., V. Maniezzo, and A. Colorni. "The ant system: Optimization by a colony of cooperating agents". *IEEE Transactions on Systems, Man, and Cybernetics–Part B*, 26(1):29–41, 1996.

34. Dorigo, Marco and Gianni Di Caro. "Ant Algorithms for Discrete Optimization". *Artificial Life*, 5(2):137–172, 1999.

35. Eyckelhof, Casper Joost and Marko Snoek. "Ant Systems for a Dynamic TSP". *Lecture Notes In Computer Science*, 2463:88–99, 2002. URL `http://portal.acm.org/citation.cfm?id=646686.702803`.

36. Fischer, M. L. and L. N. Jaikumar. "A Generalized Assignment Heuristic for the Large-Scale Vehicle Routing". *Networks*, 11:109–124, 1981.

37. Gambardella, Luca M. and Marco Dorigo. "Ant-Q: A Reinforcement Learning approach to the traveling salesman problem". *Proceedings of ML-95, Twelfth International Conference on Machine Learning*, 252–260, 1995.

38. Gambardella, Luca Maria and Marco Dorigo. "An Ant Colony System Hybridized with a New Local Search for the Sequential Ordering Problem". *INFORMS Journal on Computing*, 12(3):237–255, 2000.

39. Harchol-Balter, Mor, Mark Crovella, and Cristina D. Murta. "On Choosing a Task Assignment Policy for a Distributed Server System". *Computer Performance Evaluation (Tools)*, 231–242. 1998.

40. Heragu, S. S. "Facility Layout". *EJORDT*, 57(2):135–204, 1992.

41. Heragu, S. S. and A. Kusiak. "Efficient Models for the Facility Layout Problem". *EJOR*, 53:1–13, 1991.

42. IBM WebSphere. Http://www.ibm.com/software/websphere.

43. Kaelbling, Leslie Pack and Michael L. Littman. "Reinforcement Learning: A Survey".

44. Keren, A. and A. Barak. "Opportunity Cost Algorithms for Reduction of I/O and Interprocess Communication Overhead in a Computing Cluster". *IEEE Transactions on Parallel and Distributed Systems*, 14(1):39–50, 2003.

45. Kusiak, A. and S. S. Heragu. "The Facility Layout Problem". *EJOR*, 29:229–251, 1987.

46. Labrinidis, A. and N. Roussopoulos. “Update Propagation Strategies for Improving the Quality of Data on the Web”. *VLDB*. 2001.

47. Labrinidis, A. and N. Roussopoulos. “Balancing Performance and Data Freshness in Web Database Servers”. *VLDB*. 2003.

48. Li, Wen Syan, Kemal Altinas, and Mura Kantarcioglu. “On Demand Synchronization and Load Distribution for Database Grid-based Web Applications”. *Data and Knowledge Engineering*, 51(3):295–323, December 2004.

49. Li, Wen-Syan, Oliver Po, Wang-Pin Hsiung, K. S. Candan, Divyakant Agrawal, Yusuf Akca, and Kunihiro Taniguchi. “CachePortal II: Acceleration of Very Large Scale Data Center-Hosted Database-driven Web Applications”. *VLDB*, 1109–1112. 2003.

50. LINDO Systems, Inc. Http://www.lindo.com.

51. Luo, Qiong, Sailesh Krishnamurthy, C. Mohan, Hamid Pirahesh, Honguk Woo, Bruce G. Lindsay, and Jeffrey F. Naughton. “Middle-tier database caching for e-business”. *SIGMOD*, 600–611. 2002.

52. Majumdar, Ratul, Krithi Ramamritham, Ravi N. Banavar, and Kannan M. Moudgalya. “Disseminating Dynamic Data with QoS Guarantee in a Wide Area Network: A Practical Control Theoretic Approach”. *RTAS*, 510–517. 2004.

53. Mayer, Christopher. *Quality-Based Replication of Freshness-Differentiated Web Applications and Their Back-End Databases*. Ph.D. thesis, Arizona State University, 2005.

54. Michel, R. and M. Middendorf. “An island model based ant system with lookahead for the shortest supersequence problem”. *Lecture Notes In Computer Science*, 1498:692–701, 1998.

55. Mohan, C. “Caching Technologies for Web Applications”. *VLDB*. 2001.

56. Montemanni, R., L.M. Gambardella, A.E. Rizzoli, and A.V. Donati. *A new algorithm for a Dynamic Vehicle Routing Problem based on Ant Colony System*. Technical Report IDSIA-23-02, Istituto Dalle Molle di Studi sull’Intelligenza Artificiale (IDSIA), 2002.

57. Olston, Chris and Jennifer Widom. “Best-Effort Cache Synchronization with Source Cooperation”. *SIGMOD*, 73–84. 2002. URL `citeseer.nj.nec.com/olston02besteffort.html`.

58. Pape, Cécile Le, Stéphane Gançarski, and Patrick Valduriez. “Refresco: Improving Query Performance Through Freshness Control in a Database Cluster”. *CoopIS*, 174–193. 2004.

59. Pardalos, Panos M. and James V. Crouse. “A Parallel Algorithm for the Quadratic Assignment Problem”. *ACM*, 1989.

60. Pigatti, Alexandre, Marcus Poggi de Aragao, and Eduardo Uchoa. “Stabilized Branch-and-Cut-and-Price for the Generalized Assignment Problem”, October 2004.

61. Ross, G. T. and R. M. Soland. “Modeling Facility Location Problems as Generalized Assignment Problems”. *Management Science*, 24:345–357, 1977.

62. Schmidhuber, J. “A general method for multi-agent learning and incremental self-imrovement in unrestricted environments.” *Evolutionary Computation: Theory and Applications.*, 1996.

63. Schoonderwoerd, R., O. Holland, J. Bruten, and L. Rothkrantz. “Ant-based load balancing in telecommunications networks”. *Adaptive Behavior*, 5(2):169–207, 1996.

64. Strehl, Alex. “Exploration and Exploitation Strategies for the k-armed Bandit Problem”.

65. Vallamsetty, Udaykiran, Krishna Kant, and Prasant Mohapatra. “Characterization of E-Commerce Traffic”. *Intl Workshop on Advanced Issues of E-Commerce and Web-based Info Systems*, 137–144. 2002.

66. White, T., B. Pagurek, and F. Oppacher. “Connection management using adaptive mobile agents”. H.R. Arabnia (editor), *Parallel and Distributed Processing Techniques and Applications*, 802–809. 1998.

67. Widom, Jennifer. “Research Problems in Data Warehousing”. *CIKM*. 1995.

68. Wu, Ming-Chuan and Alejandro P. Buchmann. “Research Issues in Data Warehousing”. *BTW*, 61–82. 1997.

69. Zimokja, V. A. and M. I. Rubinschtein. “R&D Planning and the Generalized Assignment Problem”. *Automation and Remote Control*, 49:484–492, 1988.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
14–09–2006	Master's Thesis	Sept 2005 — Sept 2006

4. TITLE AND SUBTITLE

Optimizing the Replication of
Multi-Quality Web Applications
Using ACO and WoLF

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Dressler, Judson C, 2Lt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCS/ENG/06-05

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Thomas A. Clark
AFRL/IFSE
525 Brooks Road Rome, New York 13441-4505
(315) 330-2904 DSN 587-2904
MAJCOM: Air Force Materiel Command

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This thesis presents the adaptation of Ant Colony Optimization to a new NP-hard problem involving the replication of multi-quality database-driven web applications (DAs) by a large application service provider (ASP). The ASP must assign DA replicas to its network of heterogeneous servers so that user demand is satisfied and replica update loads are minimized. The algorithm proposed, AntDA, for solving this problem is novel in several respects: ants traverse a bipartite graph in both directions as they construct solutions, pheromone is used for traversing from one side of the bipartite graph to the other and back again, heuristic edge values change as ants construct solutions, and ants may sometimes produce infeasible solutions. Experiments show that AntDA outperforms several other solution methods, but there was room for improvement in the convergence rates of the ants. Therefore, in an attempt to achieve the goals of faster convergence and better solution values for larger problems, AntDA was combined with the variable-step policy hill-climbing algorithm called Win or Learn Fast (WoLF). In experimentation, the addition of this learning algorithm in AntDA provided for faster convergence while outperforming other solution methods.

15. SUBJECT TERMS

Ant Colony Optimization, Learning Algorithms, Win or Learn Fast, Policy Hill Climbing, Quality-Sensitive DA Replication Problem

16. SECURITY CLASSIFICATION OF: a. REPORT	b. ABSTRACT	c. THIS PAGE	17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
U	U	U	UU	77	Christopher B. Mayer, Major, USAF
					19b. TELEPHONE NUMBER *(include area code)* (937) 785-3636 x4542christopher.mayer@afit.edu